Probabilistic and Differentiable Programming in Scientific Simulators

Atılım Güneş Baydin

Joint work with Lukas Heinrich, Wahid Bhimji, Barak Pearlmutter, Don Syme, Philip Torr, Kyle Cranmer, Frank Wood and others

Collège de France
29 June 2022

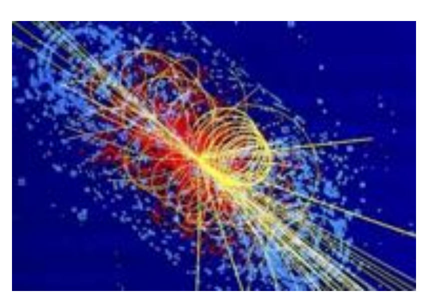
Outline

- **Probabilistic programming** and (scientific) simulators
    - Etalumis: existing simulators as probabilistic programs
    - Extensions: replacing the simulator entirely
- **Differentiable programming** and simulators
    - When autodiff is not feasible
    - When autodiff is feasible
- DiffSharp, recent work with Don Syme
- Community
    - MODE collaboration (CERN)
    - MIAPbP event (Munich)

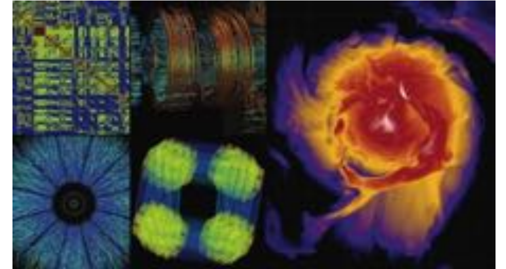
Probabilistic programming and scientific simulators

Simulation and physical sciences

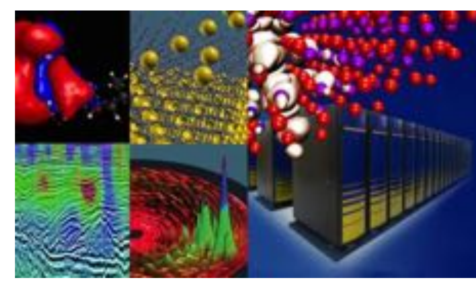
Computational models and simulation are key to scientific advance at all scales



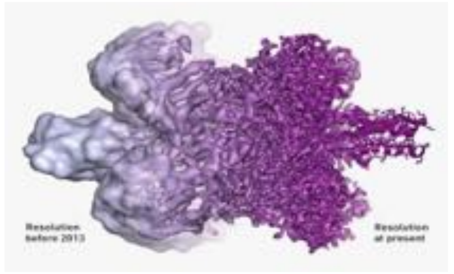
Particle physics



Nuclear physics



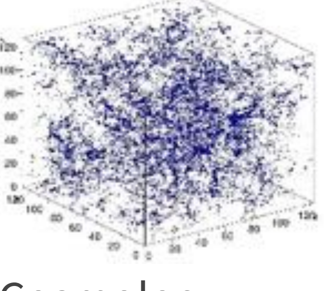
Material design



Drug discovery

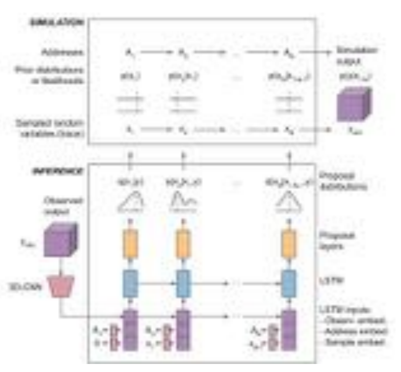
Weather

Climate science

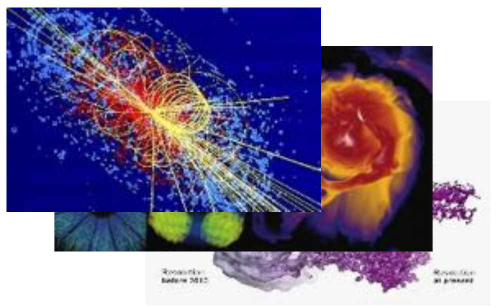


Cosmology

Introducing a new way to use existing simulators

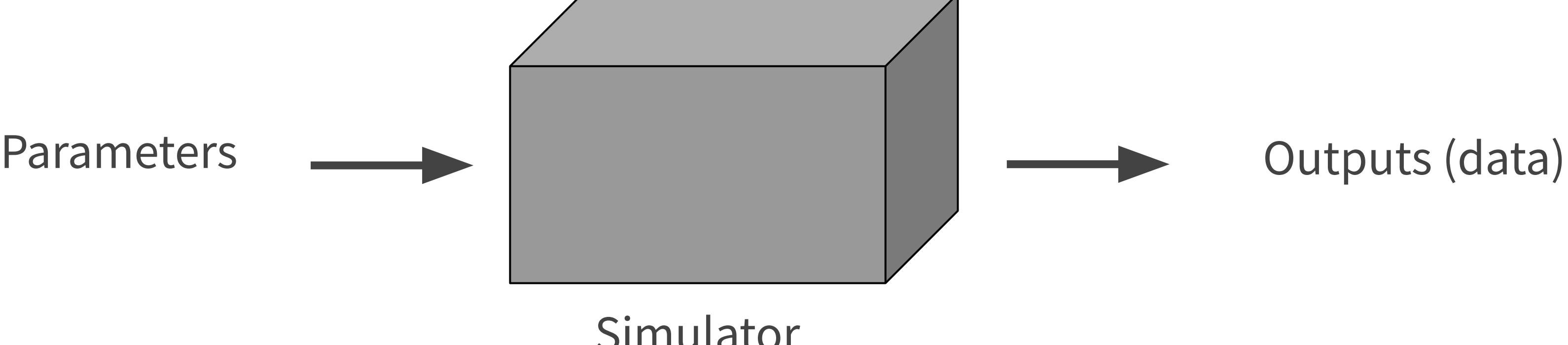


Probabilistic programming

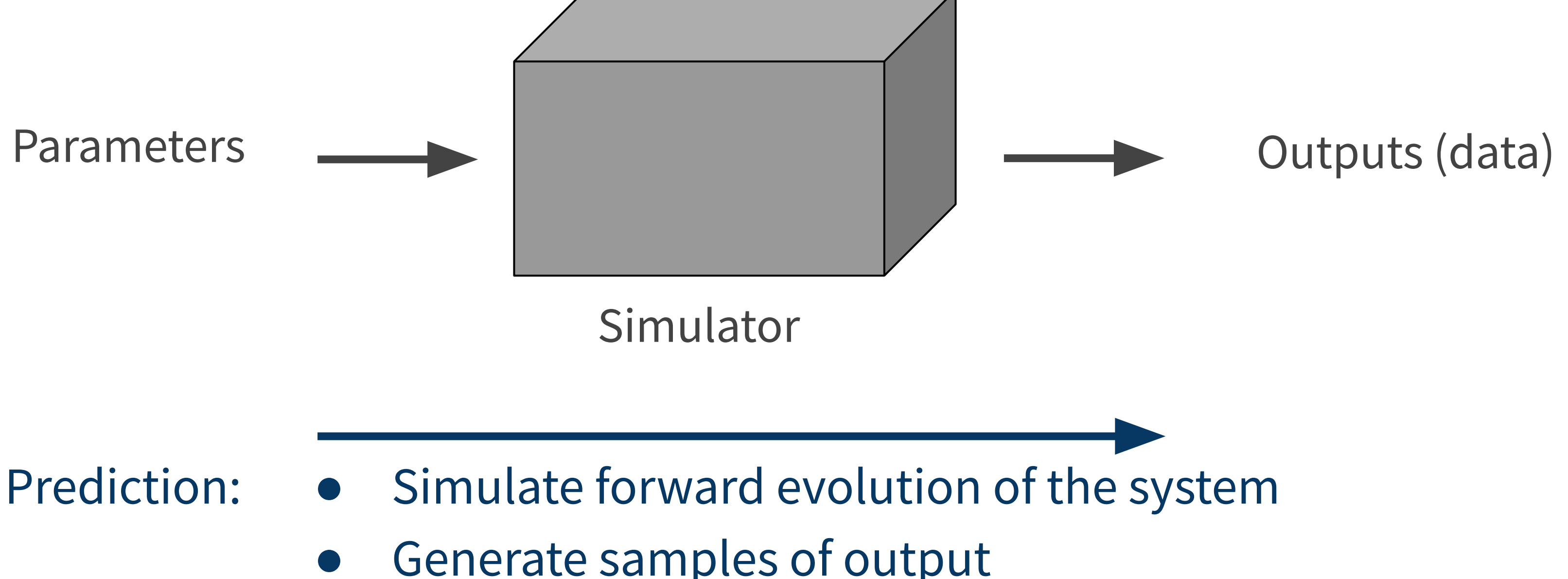


Simulation

Simulators



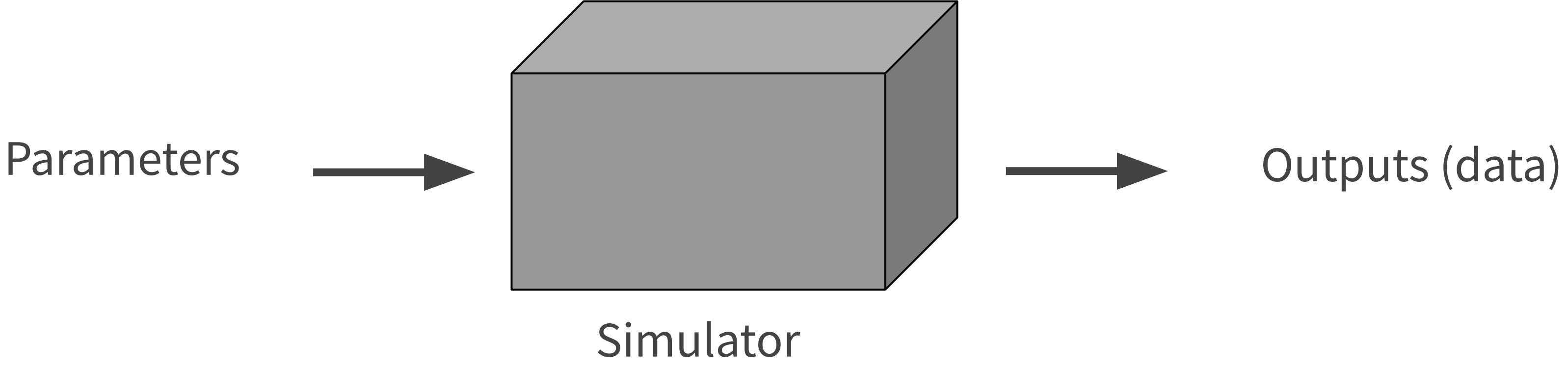
Simulators



Prediction:

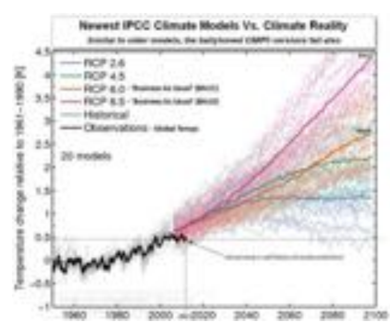
- Simulate forward evolution of the system
- Generate samples of output

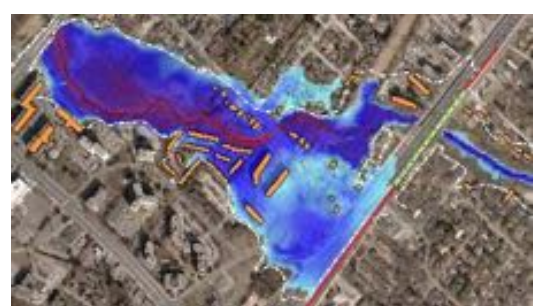
Simulators

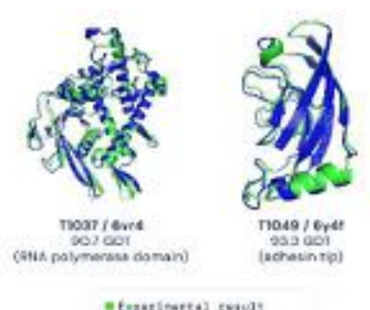


Prediction:

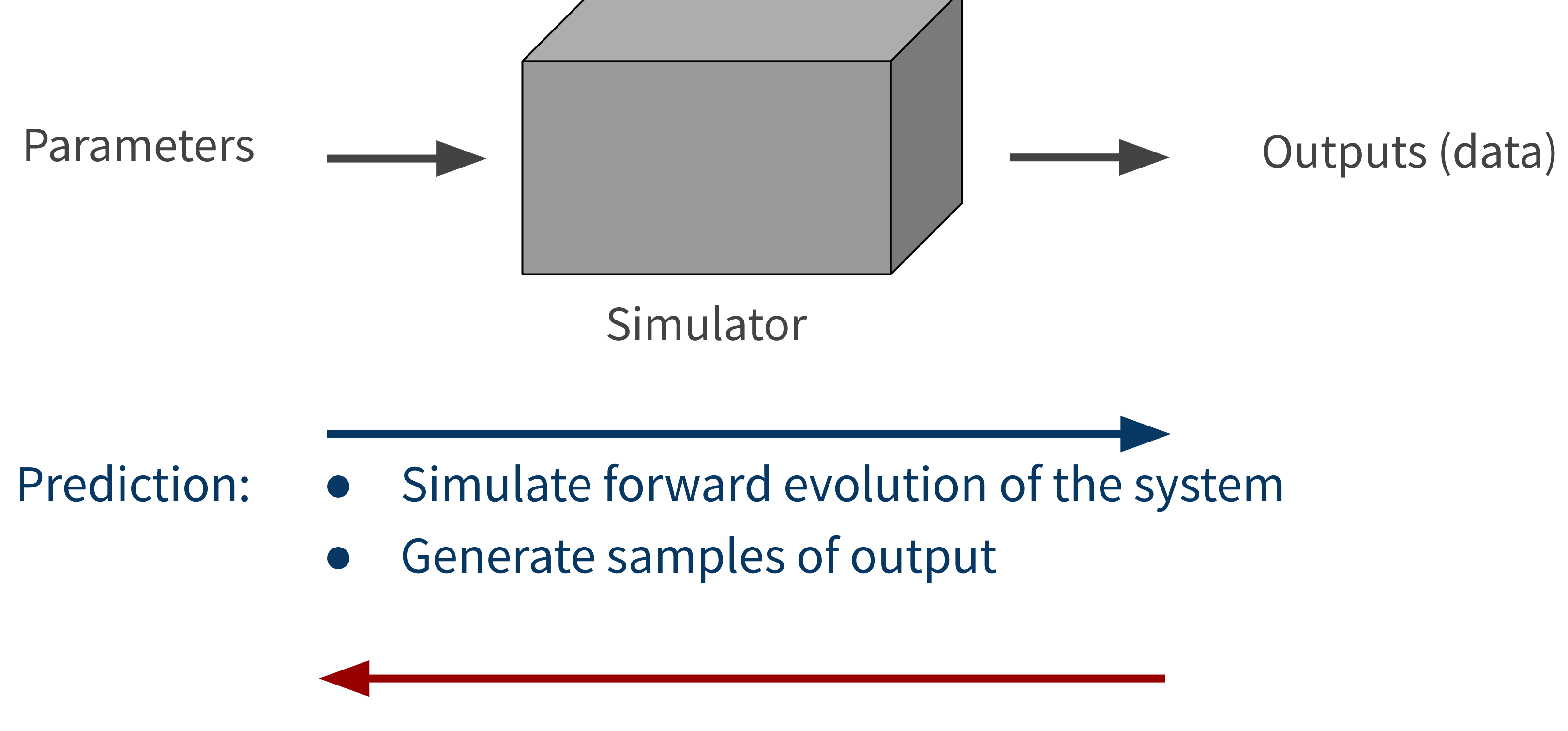
- Simulate forward evolution of the system
- Generate samples of output



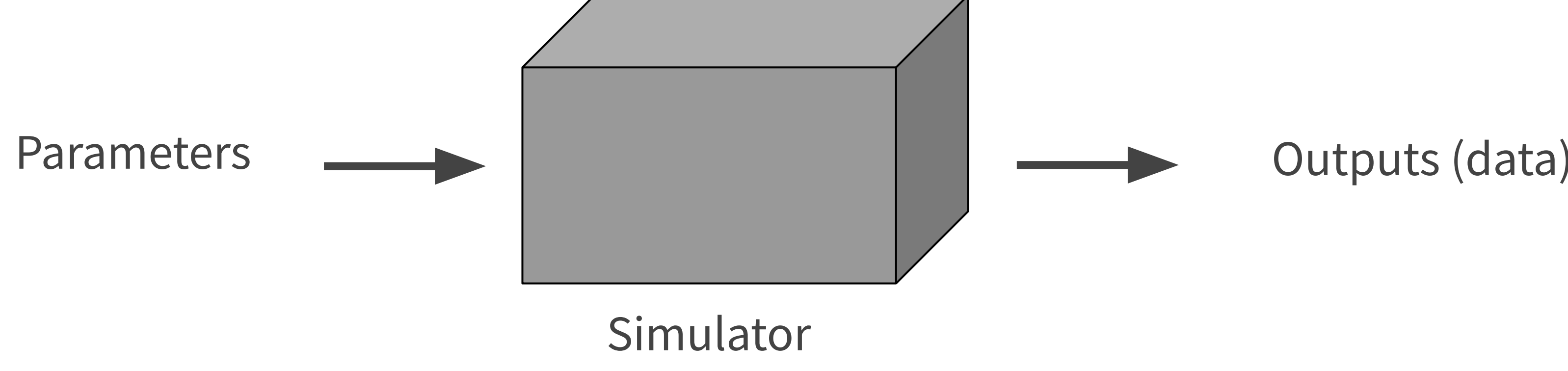




Simulators



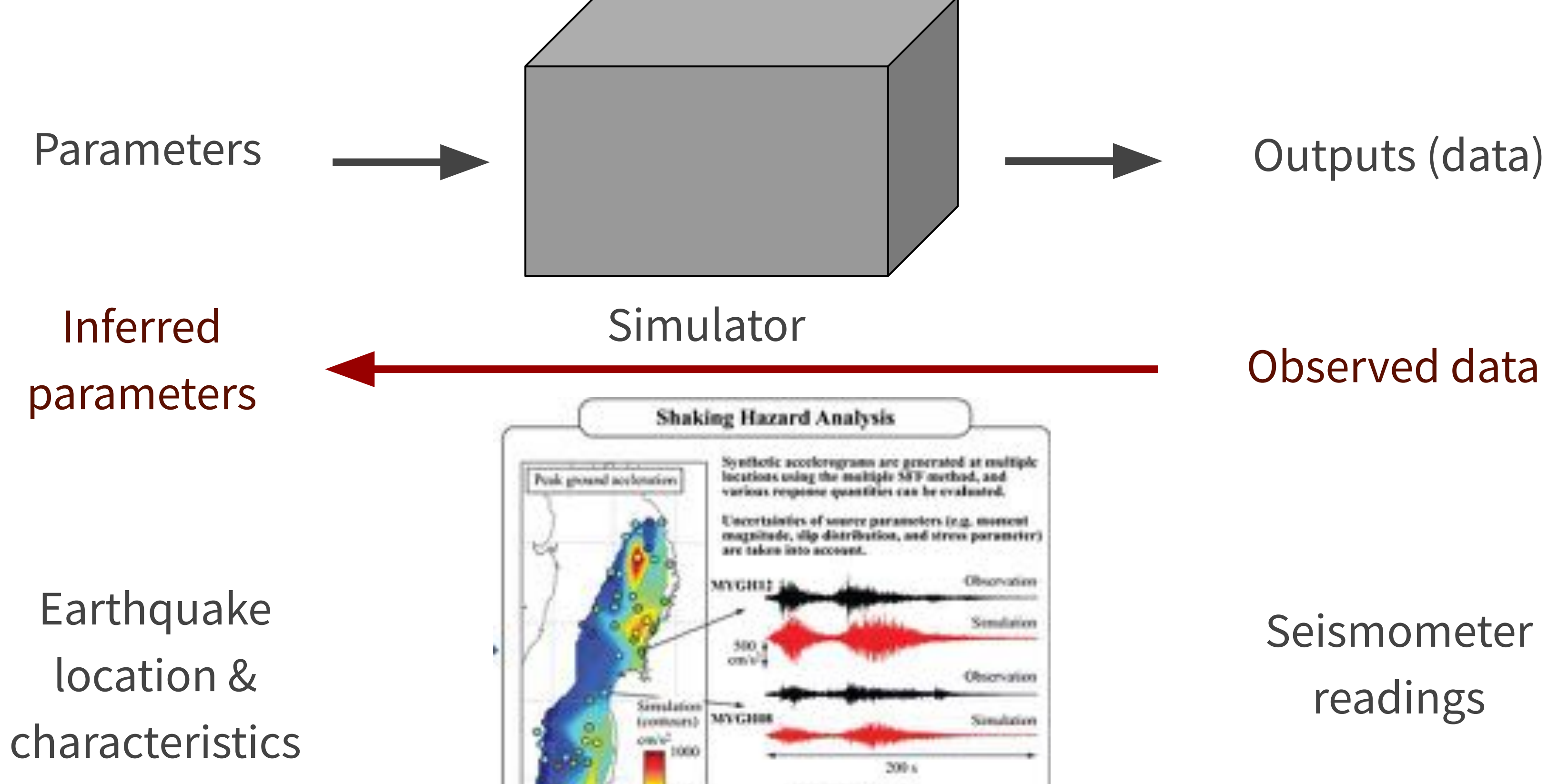
Simulators



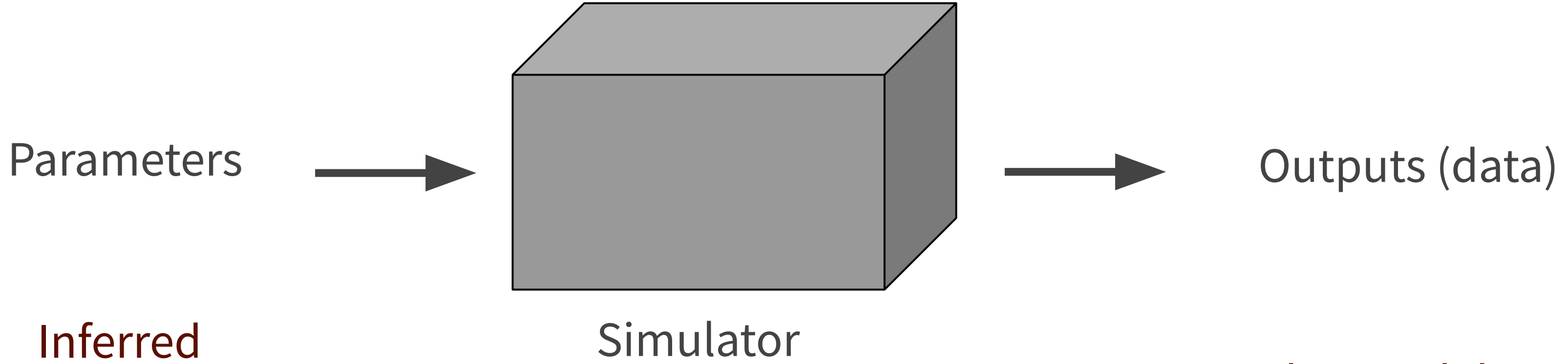
Prediction:
- Simulate forward evolution of the system
- Generate samples of output

Inference:
- Find parameters that can produce (explain) observed data
- Inverse problem
- Often a manual process

Simulators



Simulators



Parameters

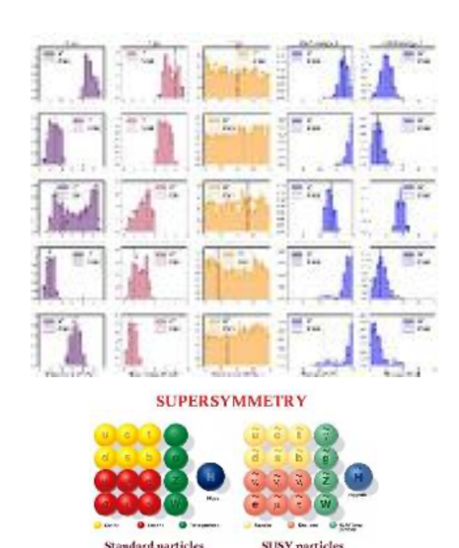
Outputs (data)

Simulator

Inferred parameters

Observed data

Event analyses & new particle discoveries



First Stable Beams

ATLAS EXPERIMENT

proton-proton collisions at 13 TeV

Particle detector readings

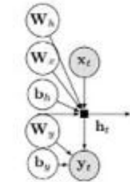
Inverting a simulator

Parameters → Simulator → Outputs (data)

Inferred parameters ← Observed data

Probabilistic programming is the perfect tool for inference

- write ***programs that define probabilistic models***
- run automated Bayesian ***inference of parameters conditioned on observed outputs*** (data)



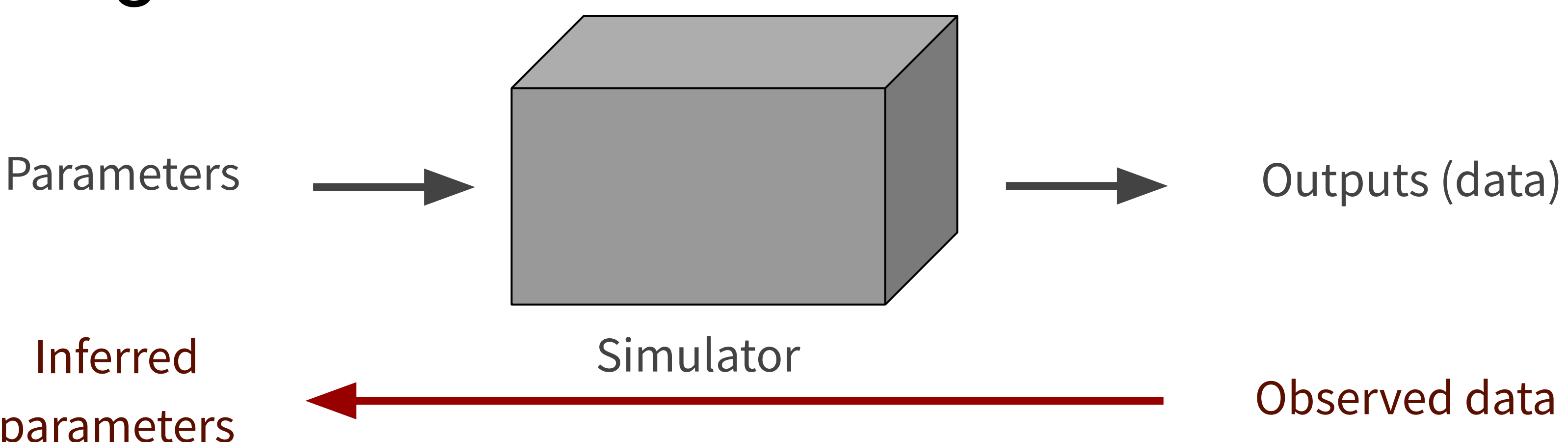
```
1  def rnn_cell(hprev, xt):
2    return tf.tanh(tf.dot(hprev, Wh) + tf.dot(xt, Wx) + bh)
3
4  Wh = Normal(mu=tf.zeros([H, H]), sigma=tf.ones([H, H]))
5  Wx = Normal(mu=tf.zeros([D, H]), sigma=tf.ones([D, H]))
6  Wy = Normal(mu=tf.zeros([H, 1]), sigma=tf.ones([H, 1]))
7  bh = Normal(mu=tf.zeros(H), sigma=tf.ones(H))
8  by = Normal(mu=tf.zeros(1), sigma=tf.ones(1))
9
10 x = tf.placeholder(tf.float32, [None, D])
11 h = tf.scan(rnn_cell, x, initializer=tf.zeros(H))
12 y = Normal(mu=tf.matmul(h, Wy) + by, sigma=1.0)
```

Pyro

Edward

Stan

Inverting a simulator



Probabilistic programming is the perfect tool for inference

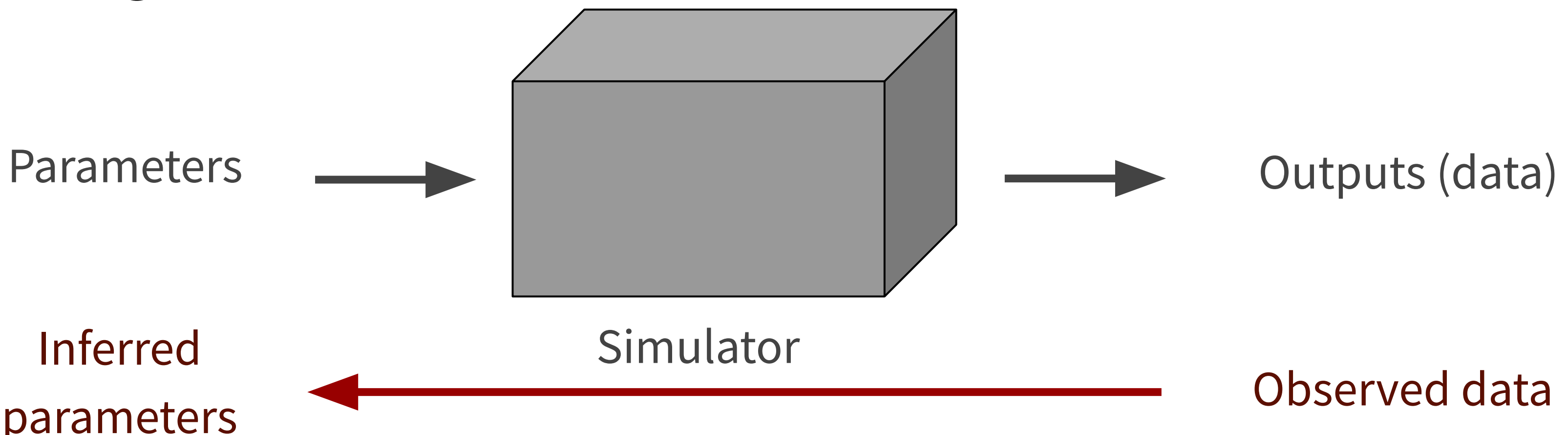
- writ
- run

con

- Somewhat limited to **small-scale problems**
- Normally requires one to **implement a probabilistic model from scratch** in the chosen language/system

Pyro Edward Stan

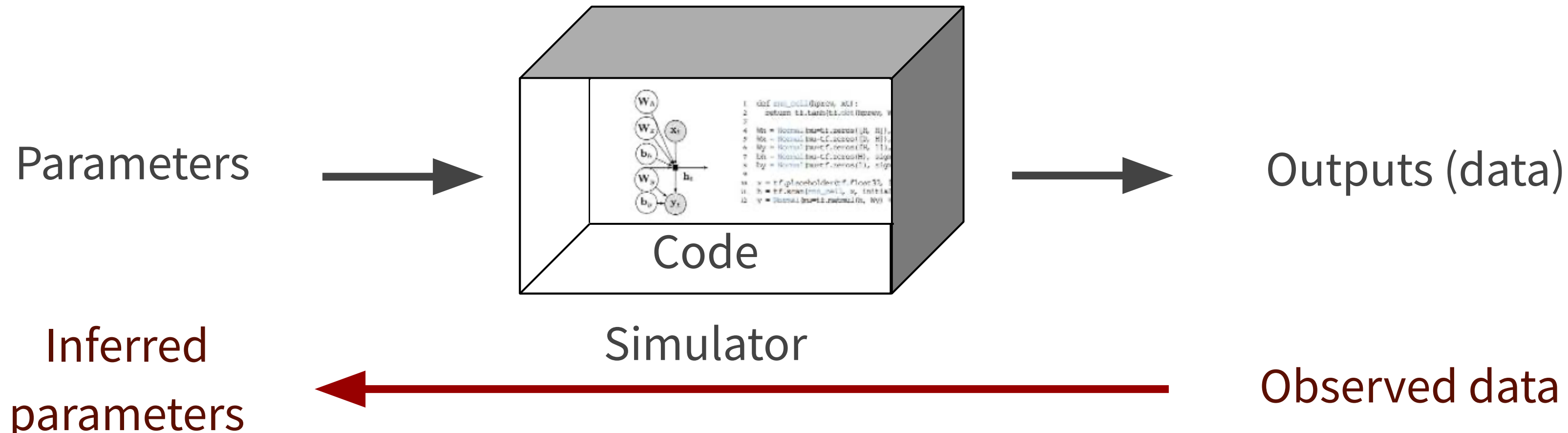
Inverting a simulator



Key idea:

Many simulators are stochastic and they define probabilistic models by sampling random numbers

Inverting a simulator

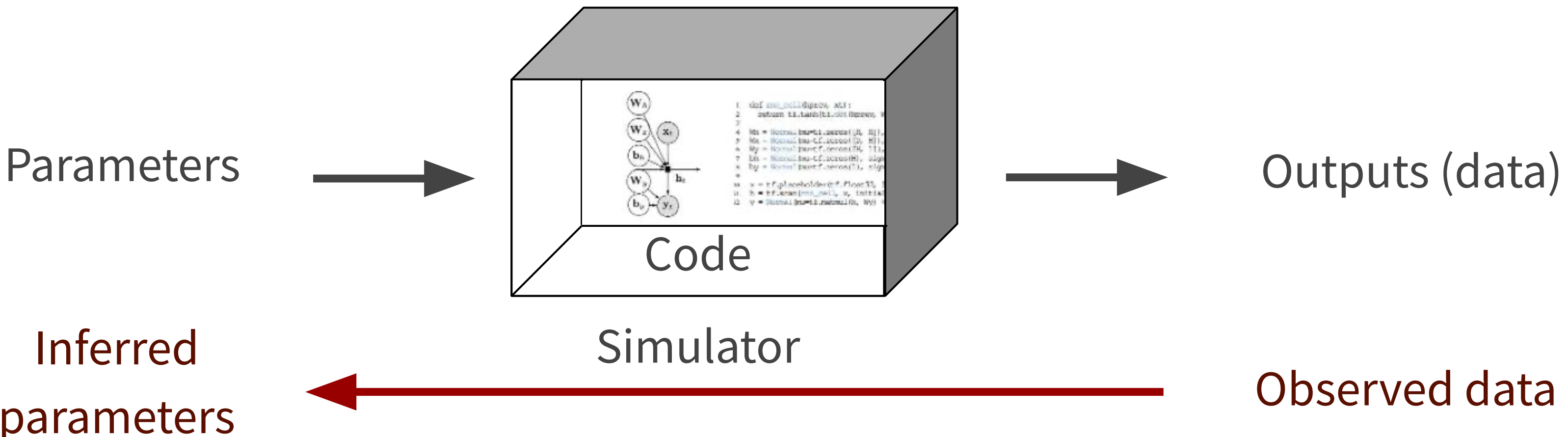


Key idea:

Many simulators are stochastic and they define probabilistic models by sampling random numbers

Simulators are probabilistic programs!

Inverting a simulator

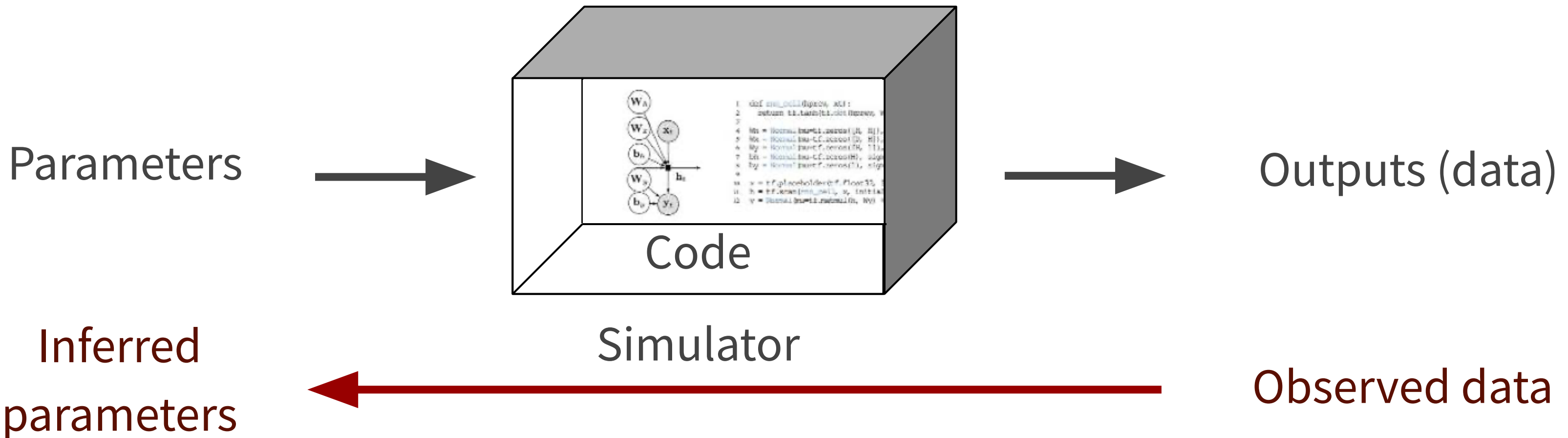


Key idea:

Many simulators are stochastic and they define probabilistic models by sampling random numbers

Simulators are probabilistic programs! (If we have the source code)

Inverting a simulator



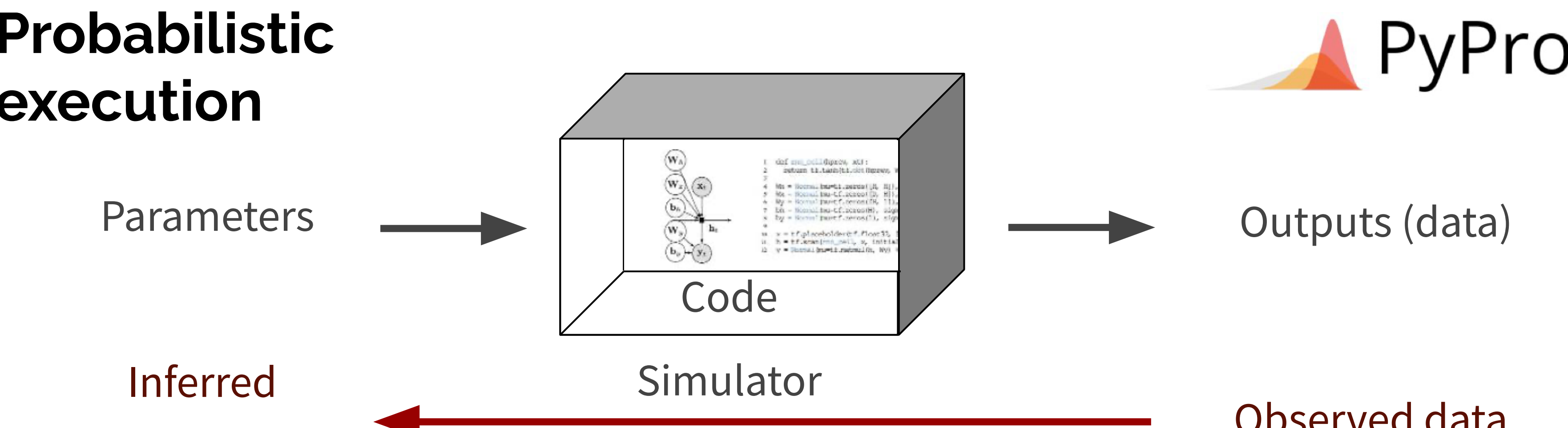
Key idea:

Many simulators are stochastic and they define probabilistic models by sampling random numbers

Simulators are probabilistic programs!

We "just" need an infrastructure to execute them probabilistically

Probabilistic execution



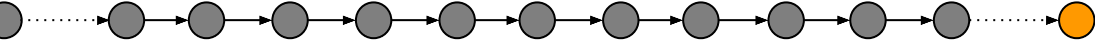
Probabilistic execution

PyProb

Parameters → Code / Simulator → Outputs (data)

Inferred parameters ← Observed data

- **Run forward & catch all random choices** (“hijack” all calls to RNG)
- Record an **execution trace**: a record of all parameters, random choices, outputs



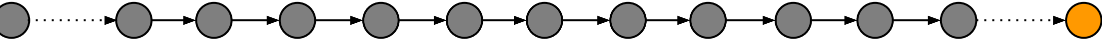
Probabilistic execution

PyProb

Parameters → Code (Simulator) → Outputs (data)

Inferred parameters ← Observed data

- **Run forward & catch all random choices** (“hijack” all calls to RNG)
- Record an **execution trace**: a record of all parameters, random choices, outputs



Probabilistic **P**rogramming e**X**ecution protocol

C++, C#, Dart, Go, Java, JavaScript, Lua, Python, Rust and others

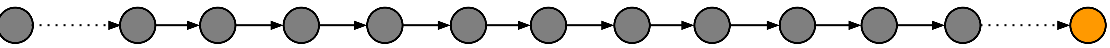
Probabilistic execution

PyProb

Parameters → Code (Simulator) → Outputs (data)

Inferred parameters ← Observed data

- **Run forward & catch all random choices** (“hijack” all calls to RNG)
- Record an **execution trace**: a record of all parameters, random choices, outputs



Probabilistic **P**rogramming e**X**ecution protocol
C++, C#, Dart, Go, Java, JavaScript, Lua, Python, Rust and others
Inspired by the Open Neural Network Exchange ONNX

Probabilistic execution

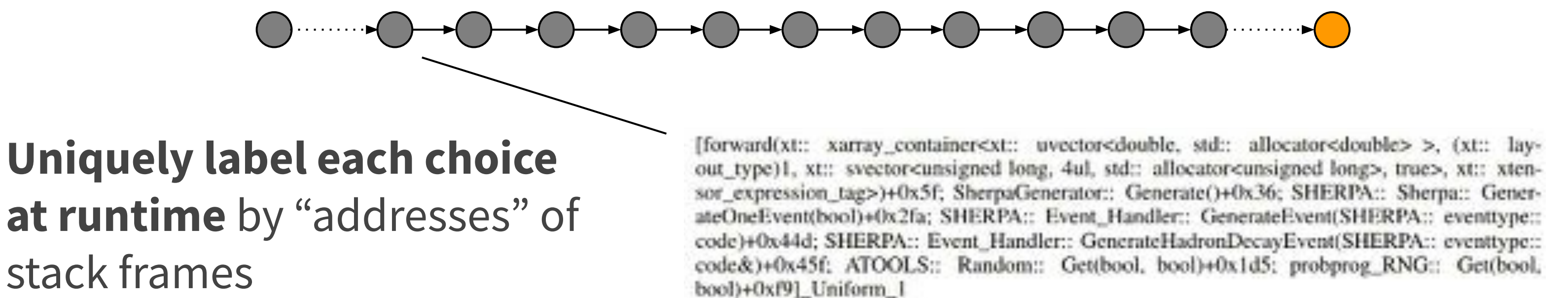
PyProb

Parameters → Code → Outputs (data)

Simulator

Inferred parameters ← Observed data

- **Run forward & catch all random choices** (“hijack” all calls to RNG)
- Record an **execution trace**: a record of all parameters, random choices, outputs



Uniquely label each choice at runtime by “addresses” of stack frames

Probabilistic execution

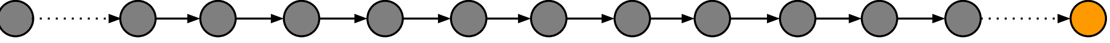
PyProb

Parameters → Code → Outputs (data)

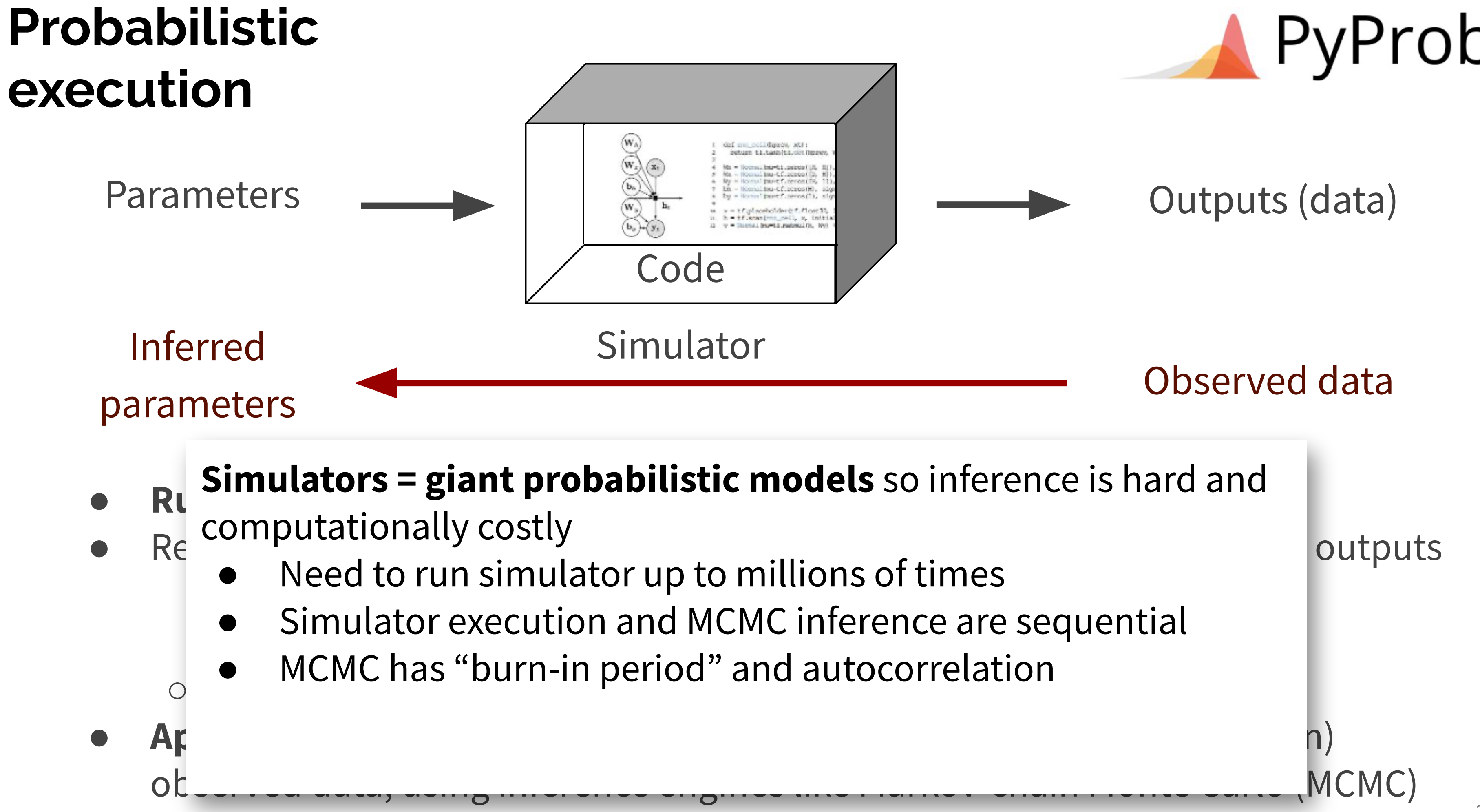
Simulator

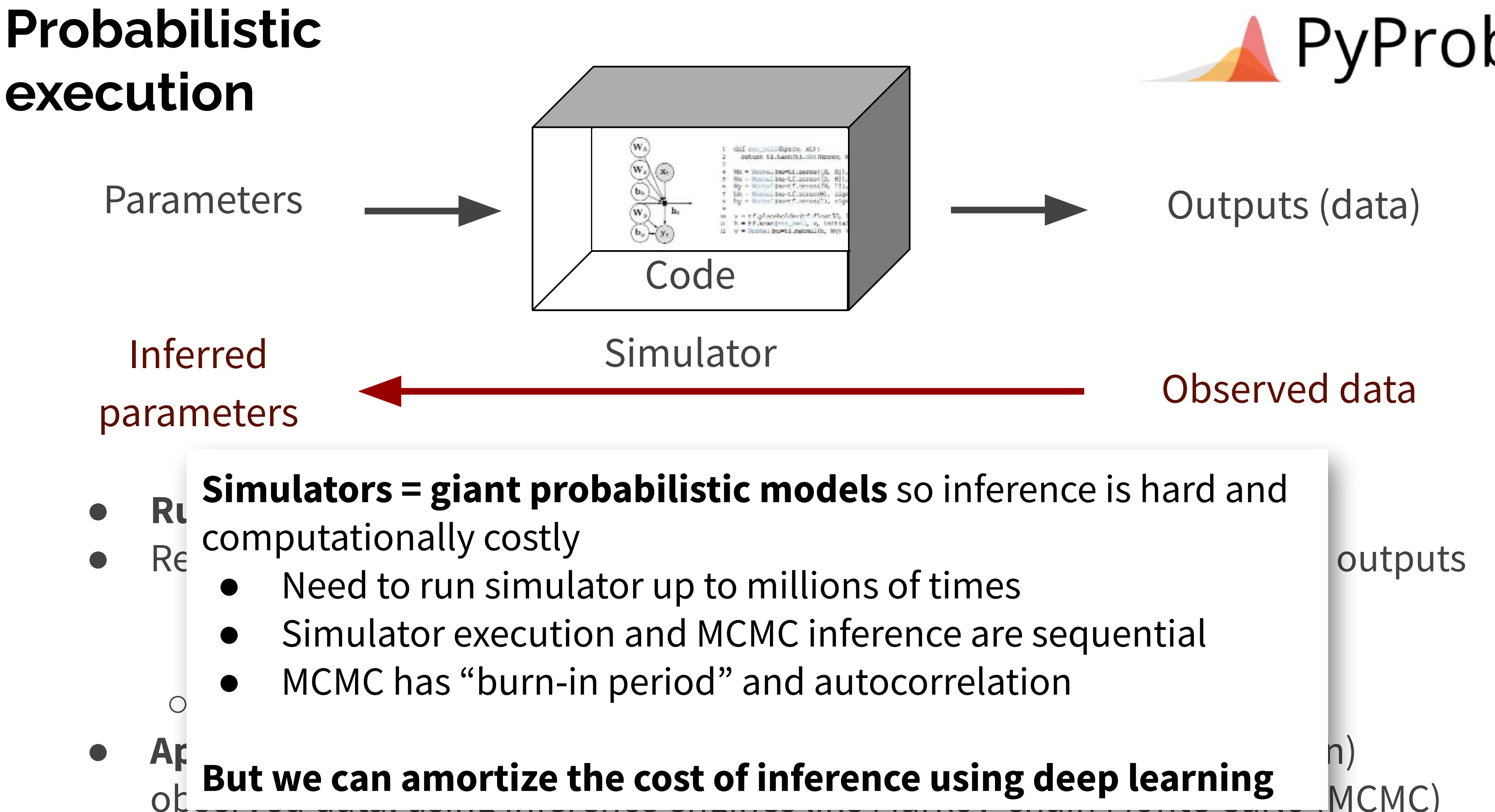
Inferred parameters ← Observed data

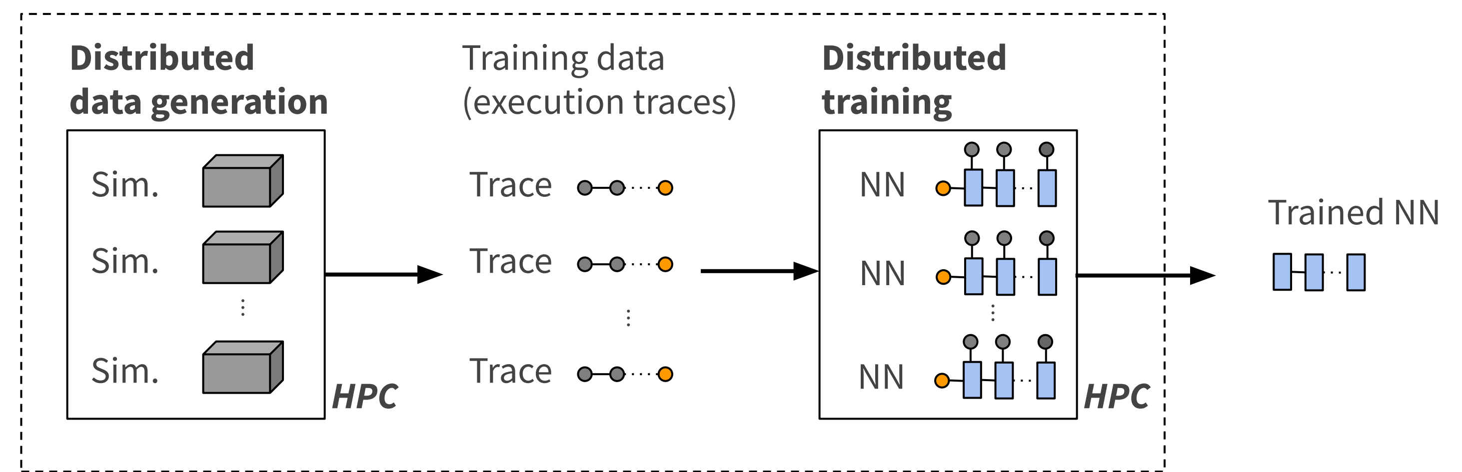
- **Run forward & catch all random choices** (“hijack” all calls to RNG)
- Record an **execution trace**: a record of all parameters, random choices, outputs



  - Conditioning: compare ***simulated output*** and ***observed data***
- **Approximate the distribution of parameters** that can produce (explain) observed data, using inference engines like Markov-chain Monte Carlo (MCMC)

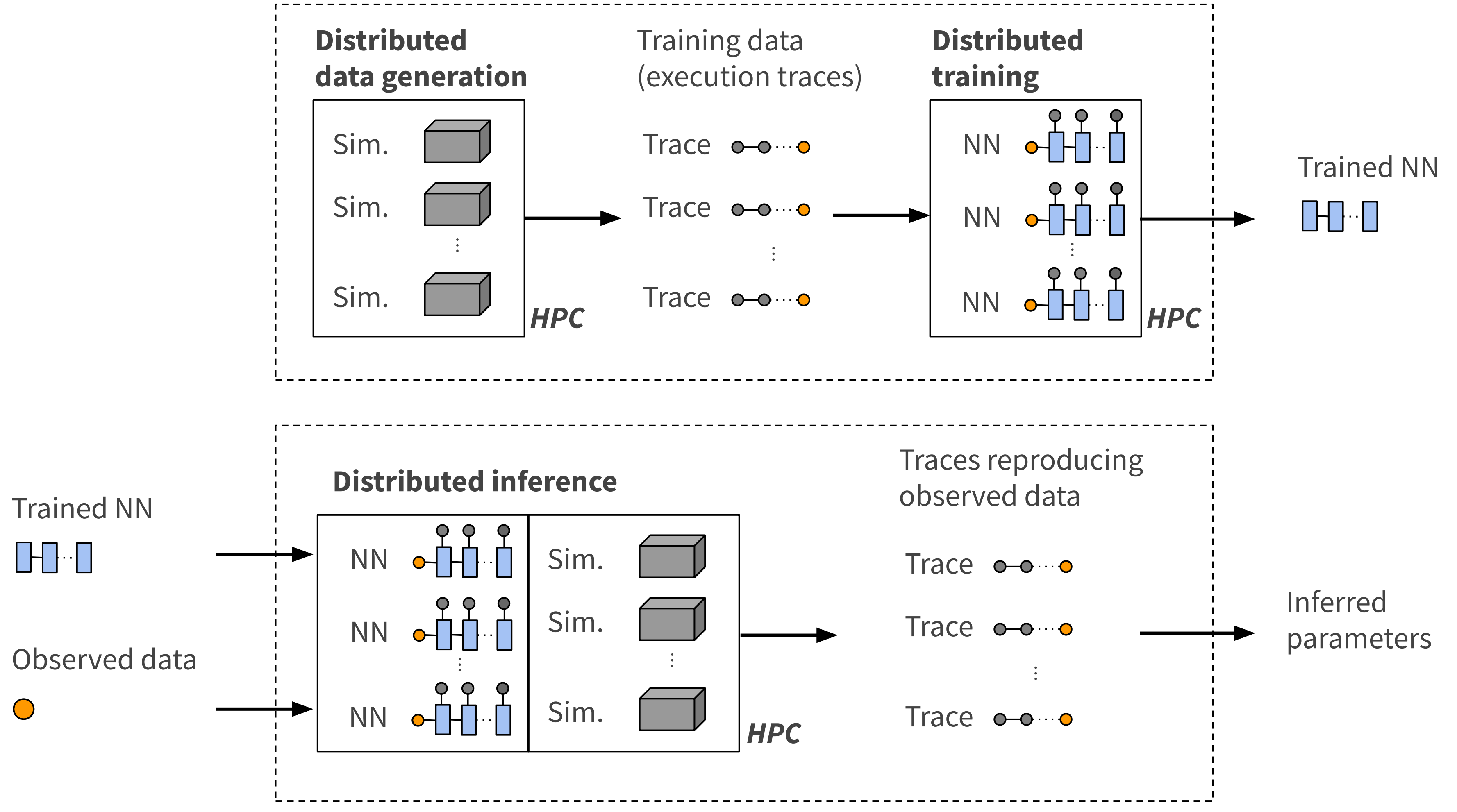

Probabilistic execution
PyProb
Parameters
Code
Outputs (data)
Simulator
Inferred parameters
Observed data
Simulators = giant probabilistic models so inference is hard and computationally costly
Need to run simulator up to millions of times
Simulator execution and MCMC inference are sequential
MCMC has "burn-in period" and autocorrelation


Probabilistic execution
PyProb
Parameters
Code
Outputs (data)
Simulator
Inferred parameters
Observed data
Ru
Re
outputs
Ap
n)
ob
MCMC)
Simulators = giant probabilistic models so inference is hard and computationally costly
Need to run simulator up to millions of times
Simulator execution and MCMC inference are sequential
MCMC has “burn-in period” and autocorrelation
But we can amortize the cost of inference using deep learning



Training (recording simulator behavior)

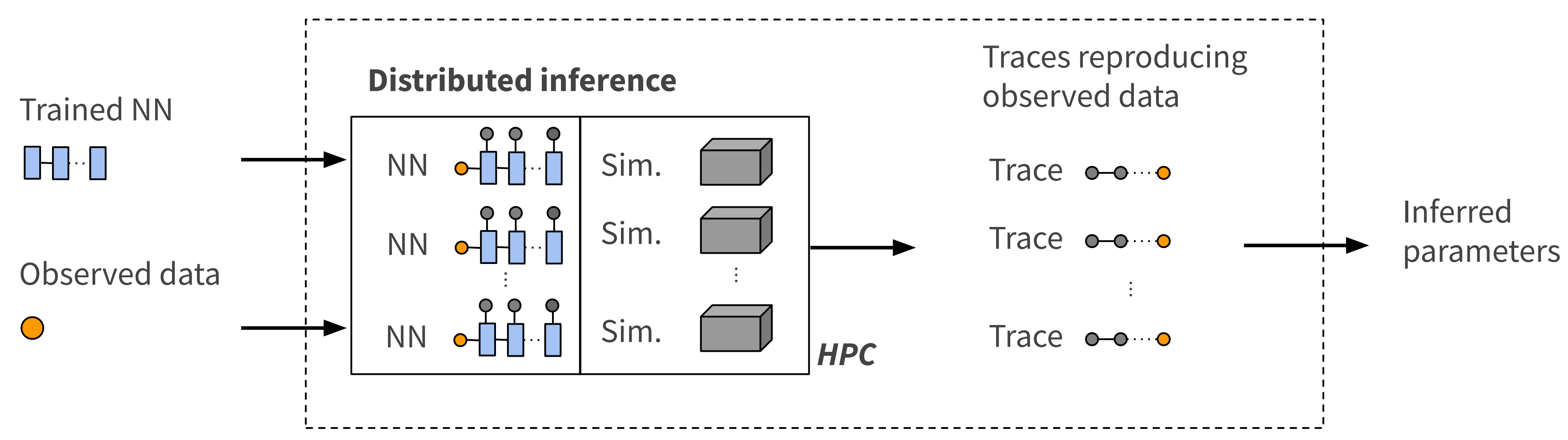
- Deep recurrent neural network learns all random choices in simulator
- Dynamic NN: grows with each simulator execution
  - Layers get created as we learn more of the simulator
  - 100s of millions of parameters in particle physics simulation
- Costly, but amortized: we need to train only once per given model



Distributed data generation
Sim.
Sim.
Sim.
HPC
Training data (execution traces)
Trace
Trace
Trace
Distributed training
NN
NN
NN
HPC
Trained NN
Trained NN
Observed data
Distributed inference
NN
NN
NN
Sim.
Sim.
Sim.
HPC
Traces reproducing observed data
Trace
Trace
Trace
Inferred parameters

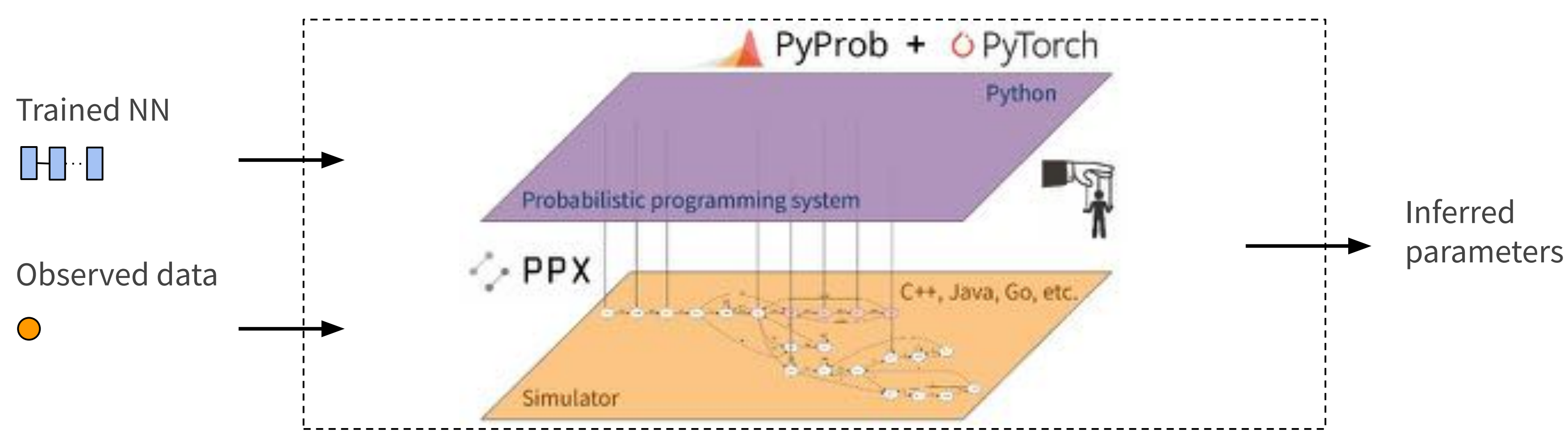
Inference (controlling simulator behavior)

- Trained deep NN makes intelligent choices given data observation
- Embarrassingly parallel distributed inference
- No burn-in period
- No autocorrelation: every sample is independent



Inference (controlling simulator behavior)

- Trained deep NN makes intelligent choices given data observation
- Embarrassingly parallel distributed inference
- No burn in period
- No autocorrelation: every sample is independent



Code infrastructure

Probabilistic programming with simulators

https://github.com/pyprob/pyprob

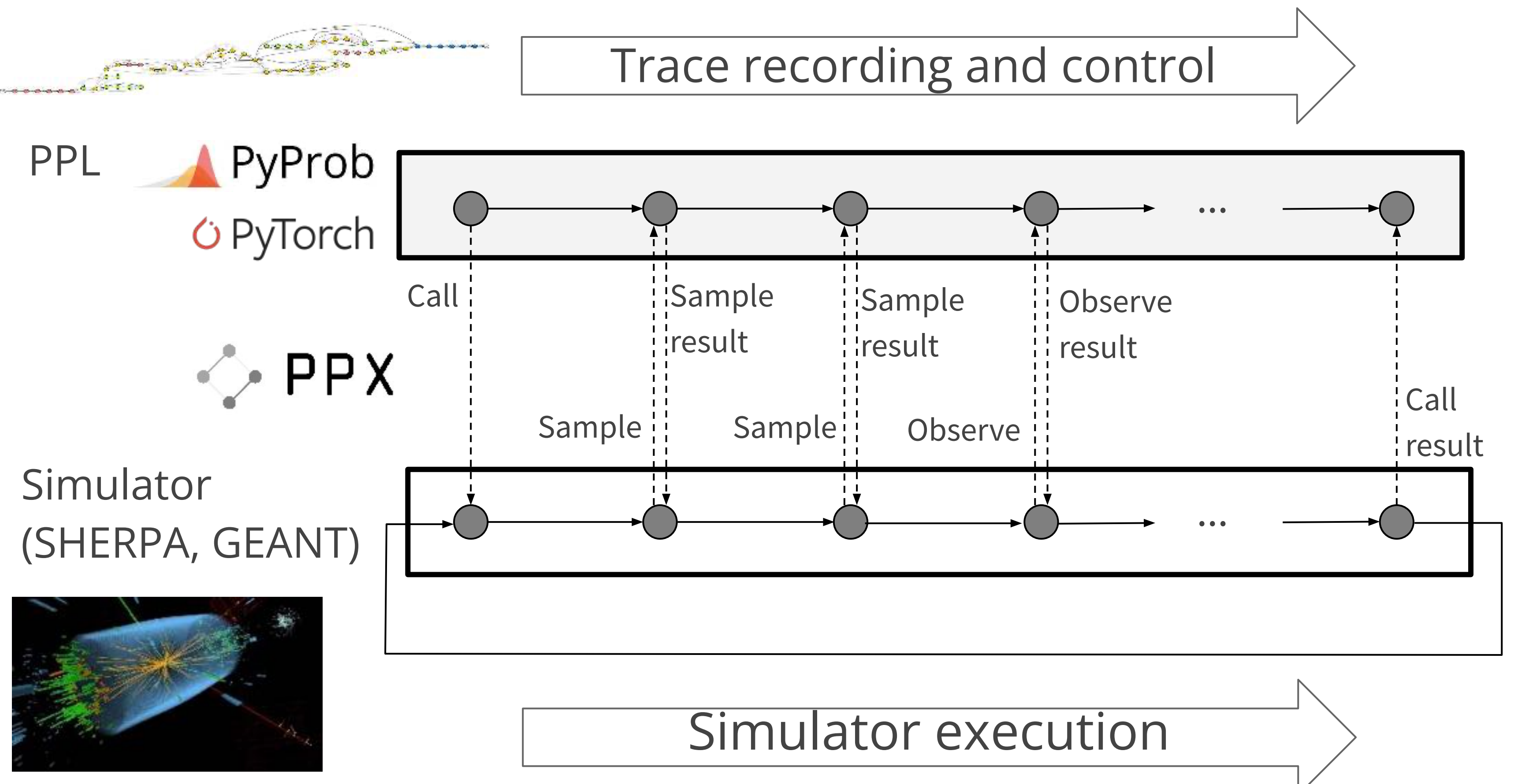
- PPL for simulators and HPC, based on PyTorch
  Distributed training and inference, efficient support for multi-TB distribution files, optimized memory usage, parallel trace processing

https://github.com/pyprob/ppx

- *P*robabilistic *P*rogramming e*X*ecution protocol
  Simulator and inference engine executed in separate processes and machines across network
  Google flatbuffers to support C++, C#, Dart, Go, Java, JavaScript, Lua, Python, Rust
  Probabilistic programming analogue to Open Neural Network Exchange (ONNX) for deep learning

Containerized workflow for HPC

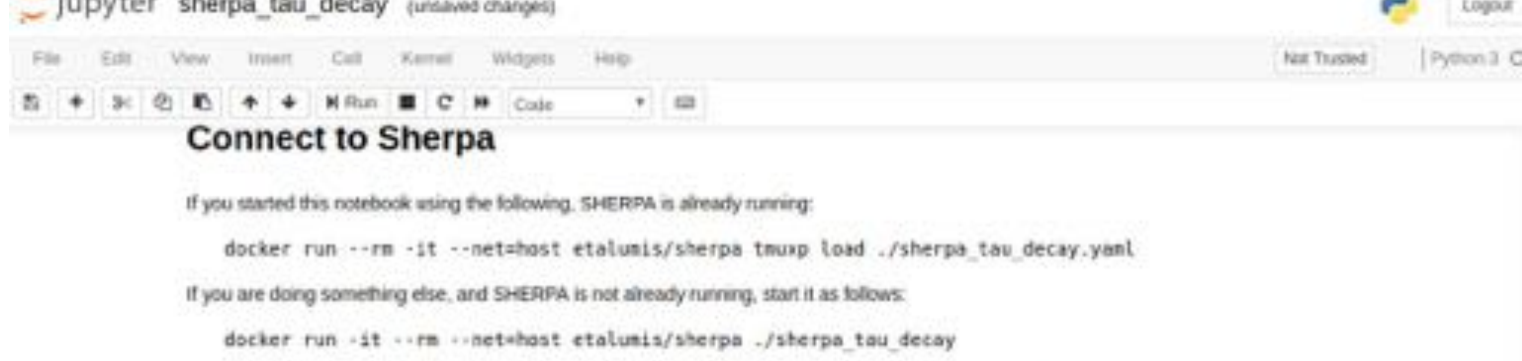
Probabilistic programming eXecution protocol



PPX in C++

```cpp
#include <pyprob_cpp.h>

// Gaussian with unkown mean
// http://www.robots.ox.ac.uk/~fwood/assets/pdf/Wood-AISTATS-2014.pdf

xt::xarray<double> forward()
{
  auto prior_mean = 1;
  auto prior_stddev = std::sqrt(5);
  auto likelihood_stddev = std::sqrt(2);

  auto prior = pyprob_cpp::distributions::Normal(prior_mean, prior_stddev);
  auto mu = pyprob_cpp::sample(prior);

  auto likelihood = pyprob_cpp::distributions::Normal(mu, likelihood_stddev);
  pyprob_cpp::observe(likelihood, "obs0");
  pyprob_cpp::observe(likelihood, "obs1");

  return mu;
}


int main(int argc, char *argv[])
{
  auto serverAddress = (argc > 1) ? argv[1] : "tcp://*:5555";
  pyprob_cpp::Model model = pyprob_cpp::Model(forward, "Gaussian with unknown mean C++");
  model.startServer(serverAddress);
  return 0;
}
```

PPX in Python



Note: by default `sherpa_tau_decay` uses the Unix domain sockets (IPC) address `ipc://@sherpa_tau_decay`. You can run it with other addresses, e.g., TCP, using: `./sherpa_tau_decay tcp://*:5555`

In [3]:
```
model = ModelRemote('ipc://@sherpa_tau_decay')
```

```
ppx (Python): zmq.REQ socket connecting to server ipc://@sherpa_tau_decay
ppx (Python): This system         : pyprob 0.10.0
ppx (Python): Connected to system: pyprob_cpp 0.1.3 (master:8110b71)
ppx (Python): Model name          : SHERPA tau lepton decay
```

Inspect the prior

We construct an empirical distribution of the prior, using a given number of `traces`. Beware, large number of traces can take a long time. `traces=500` gives a reasonable-looking plot.

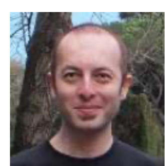
In [4]:
```
prior_dist = model.prior_distribution(num_traces=5)
```

```
Time spent  | Time remain.| Progress             | Trace | Traces/sec
0d:00:00:01 | 0d:00:00:00 | #################### | 5/5 | 2.60
```

Separate (marginalize) the distribution into distributions over momenta and channel, by mapping a function.

In [5]:
```
prior_dist_px = prior_dist.map(lambda x: float(x[0]))
prior_dist_py = prior_dist.map(lambda x: float(x[1]))
prior_dist_pz = prior_dist.map(lambda x: float(x[2]))
prior_dist_channel = prior_dist.map(lambda x: int(x[3]))
```

etalumis → | ← simulate


Atılım Güneş Baydin

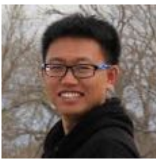
Lukas Heinrich

Wahid Bhimji

Lei Shao

Saeid Naderiparizi

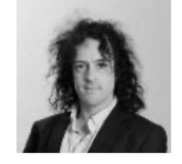
Andreas Munk


Jialin Liu

Bradley Gram-Hansen

Gilles Louppe

Lawrence Meadows


Phil Torr

Victor Lee

Prabhat

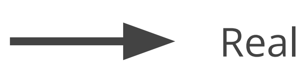
Kyle Cranmer

Frank Wood

“etalumis”
simulate

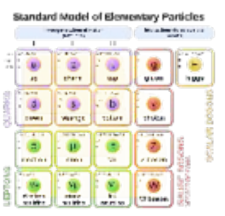
- At CERN, vast quantities of experimental data (petabytes / day) produced and compared to detailed simulations via histograms of summary statistics
  - billions of CPU hours, labor intensive, sometimes ad-hoc
- Aim: replace with automated probabilistic inference, grounded in a statistical framework

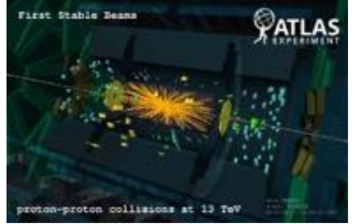
Physics



Real data

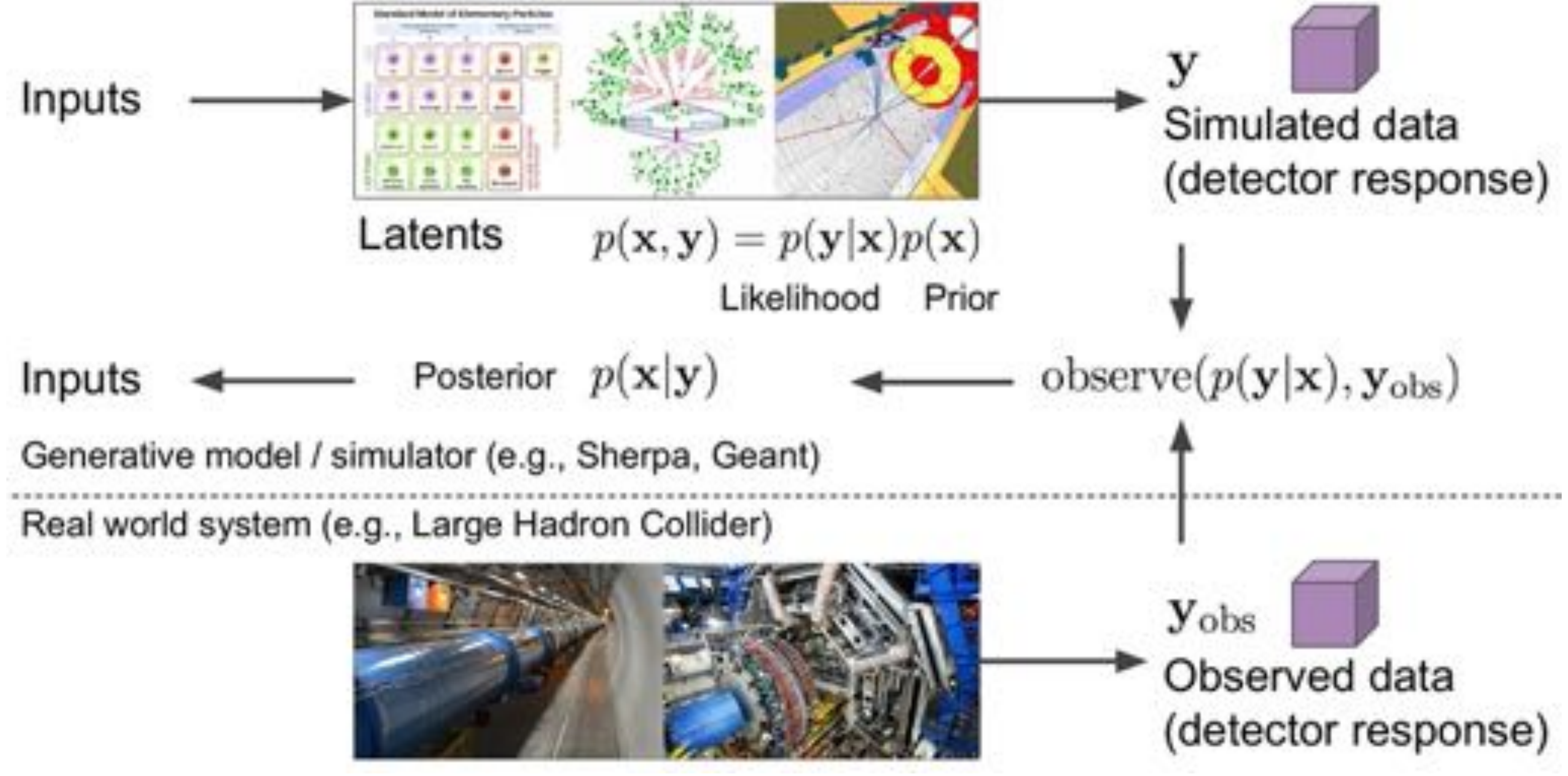
Simulation





Simulated data

Baydin et al. “Etalumis: Bringing Probabilistic Programming to Scientific Simulators at Scale” **SC19**

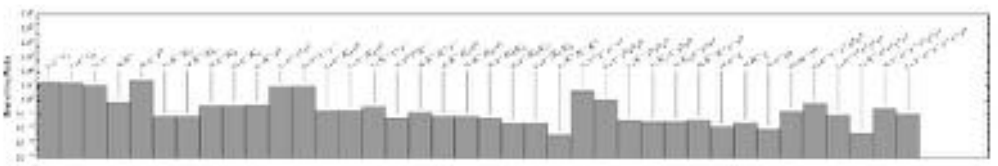


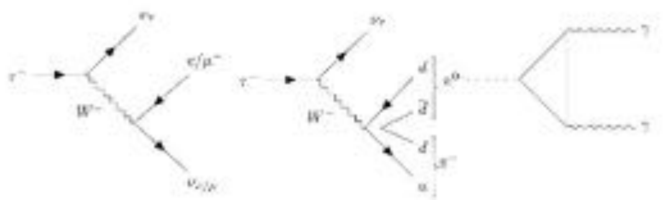
Baydin, Heinrich, Bhimji, Shao, Naderiparizi, Munk, Liu, Gram-Hansen, Louppe, Meadows, Torr, Lee, Prabhat, Cranmer, Wood.
NeurIPS 2019

“etalumis”
simulate

- Supercomputing-scale probabilistic programing ~ 25,000 latent variables; 32,768 CPU cores
- Sherpa: a state-of-the-art **simulator of the Standard Model** of particle physics, ~1M lines of C++
- Analyzing **tau lepton decay** (a key ingredient in describing the Higgs boson)
- Largest-scale PyTorch MPI (128k minibatch size)
- **First tractable Bayesian inference for LHC physics**

Cori supercomputer, Lawrence Berkeley Lab
2,388 Haswell nodes (32 cores per node)
9,688 KNL nodes (68 cores per node)



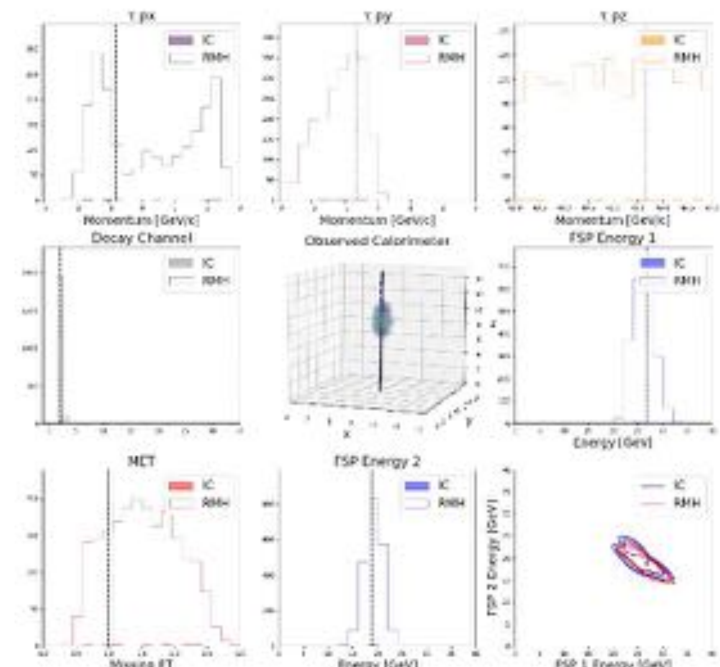


Best Paper Finalist at SC19 (supercomputing)

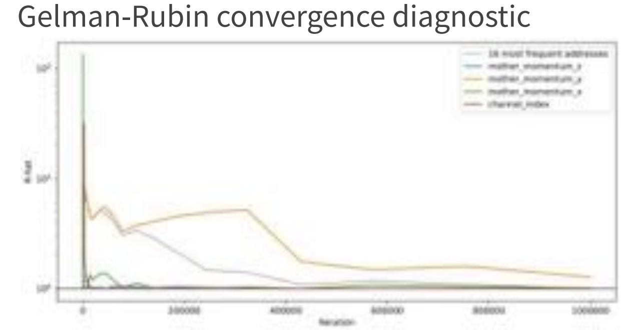
Baydin et al. “Etalumis: Bringing Probabilistic Programming to Scientific Simulators at Scale” **SC19**

Inference results

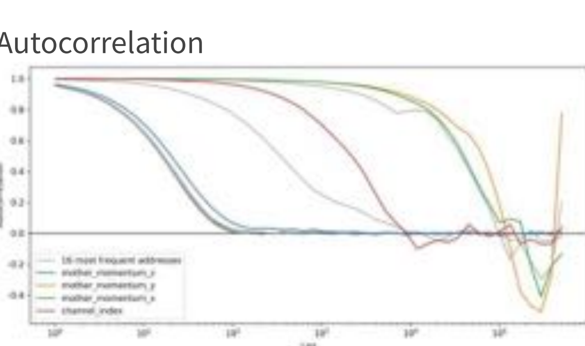
- Achieved MCMC (RMH) “ground truth”
- **First tractable Bayesian inference for LHC physics**
  - Posterior over full latent space of the Standard Model (>25k latent variables)
  - Autocorrelation typically around 10^5 steps
  - MCMC: 115 hours
- Amortized inference (IC) closely matches MCMC (RMH)
  - No autocorrelation, embarrassingly parallel
  - IC: 30 minutes



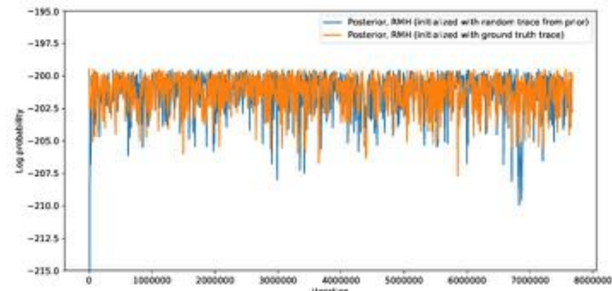
Gelman-Rubin convergence diagnostic

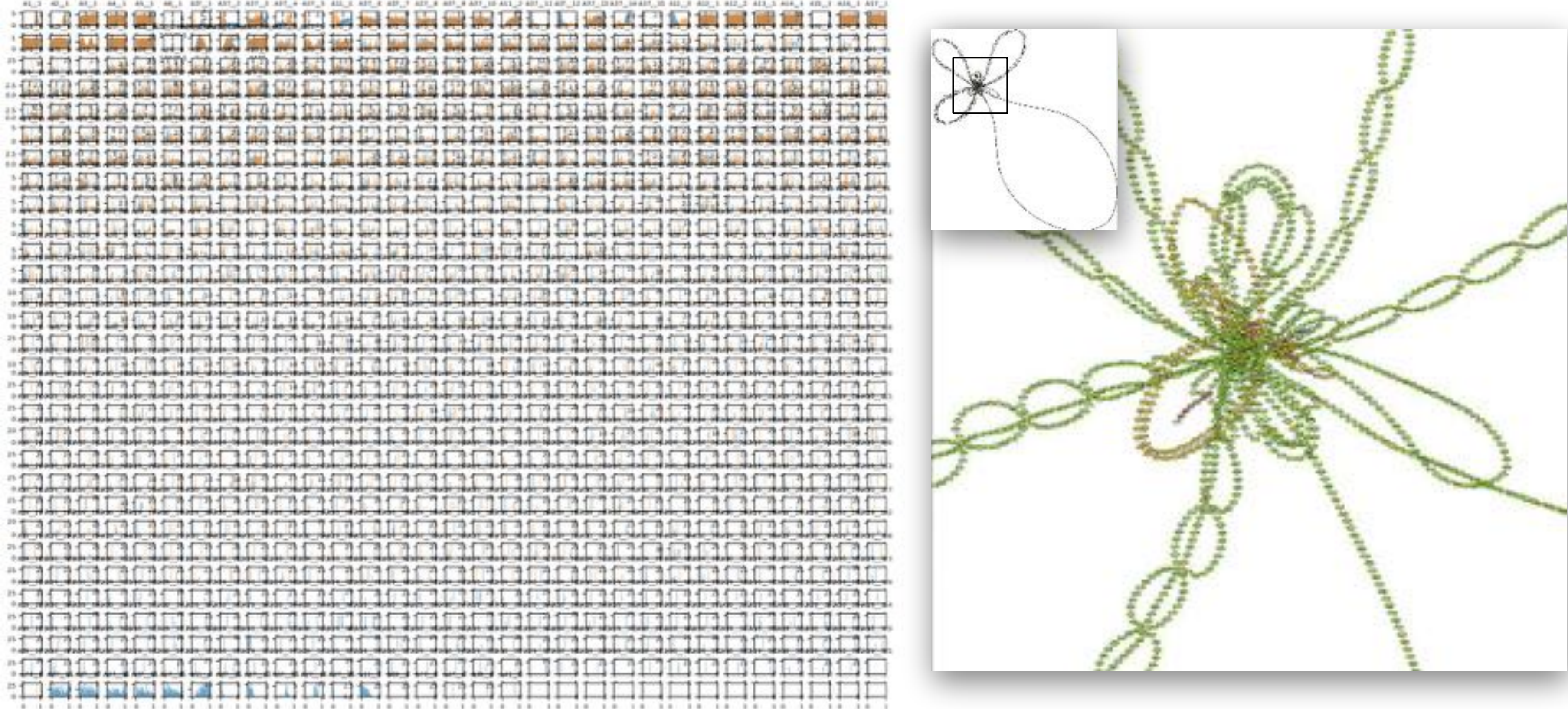


Autocorrelation



Trace log-probability

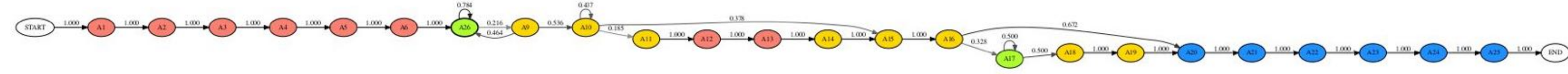




Etalumis gives access to all latent variables: allows answering *any* model-based question

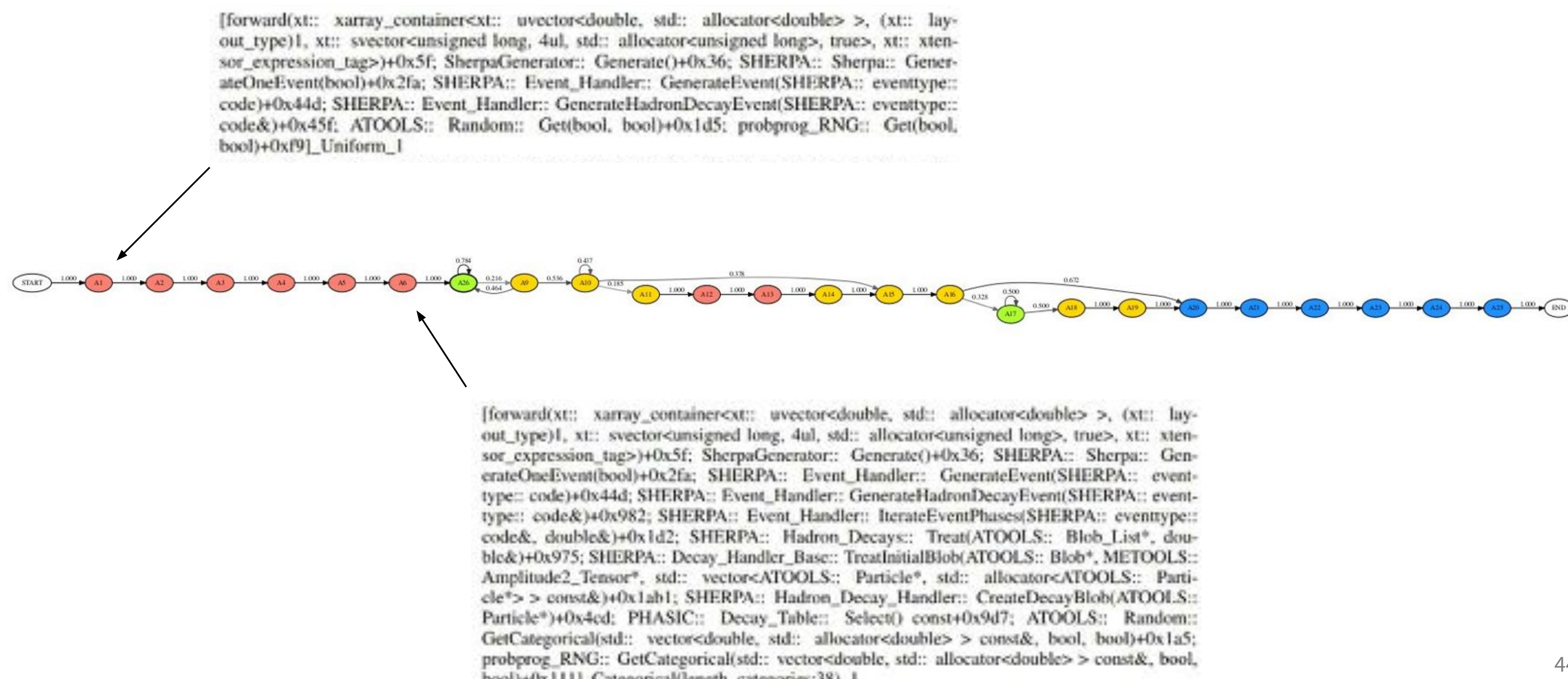
Interpretability

Latent probabilistic structure of **10** most frequent trace types



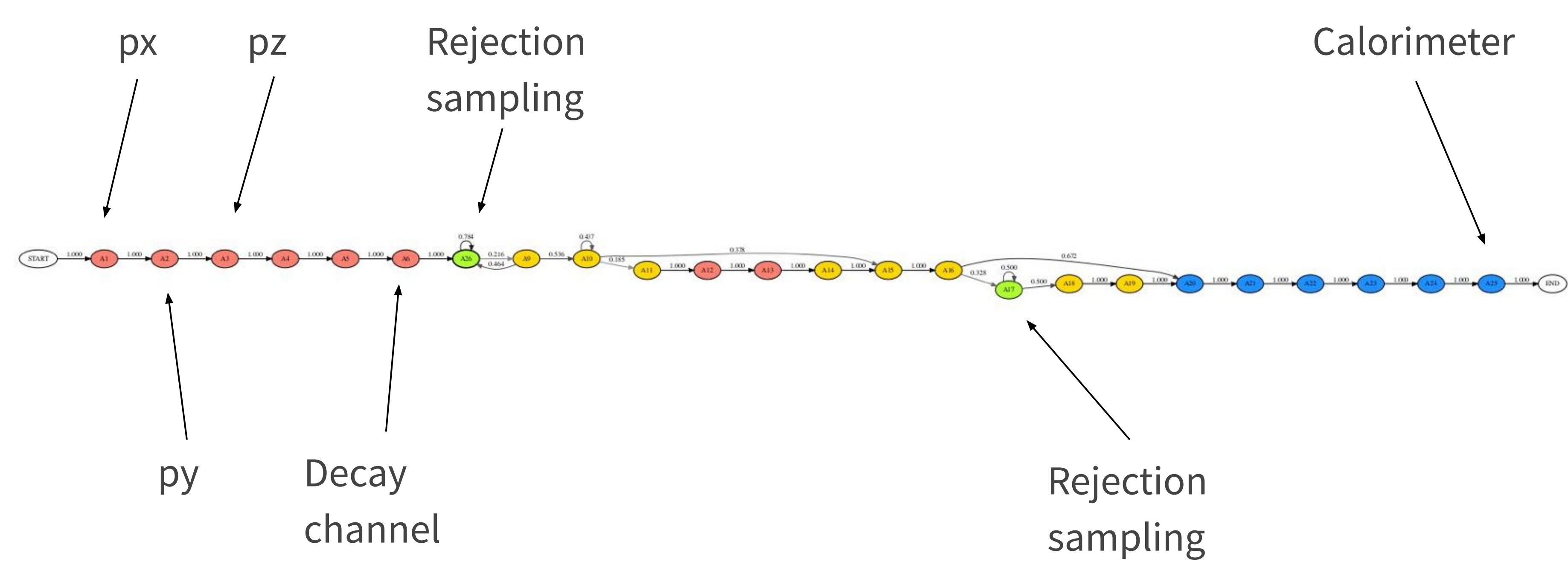
Interpretability

Latent probabilistic structure of **10** most frequent trace types



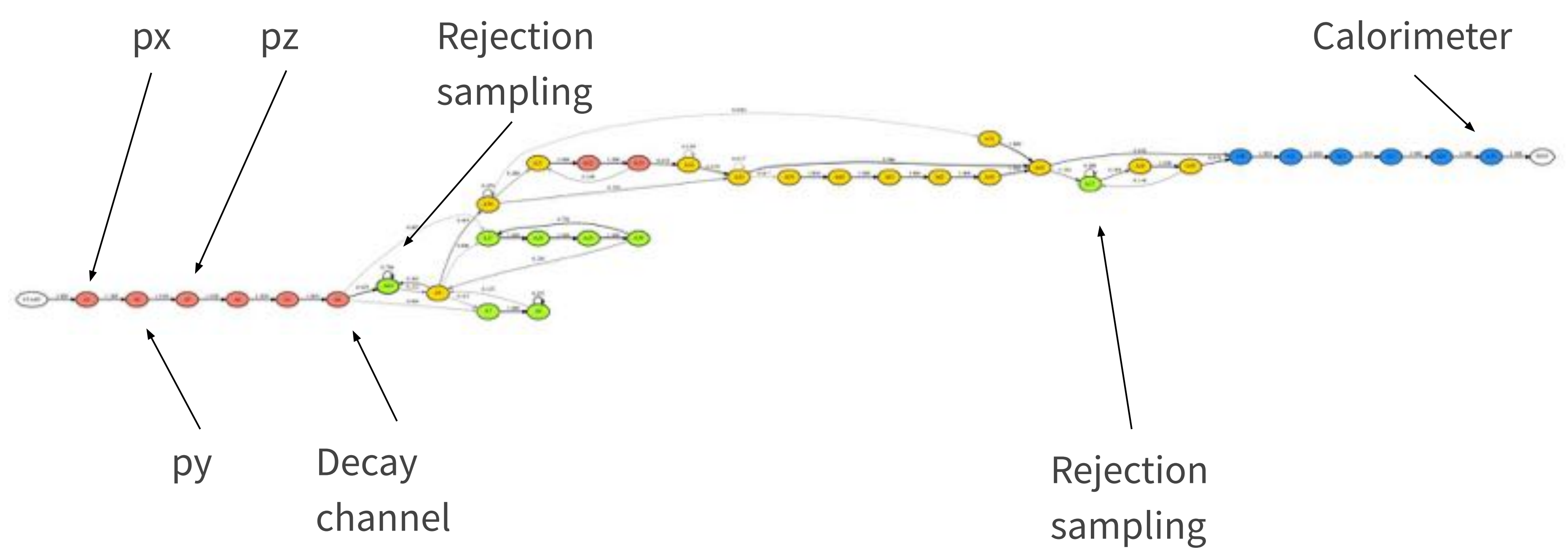
Interpretability

Latent probabilistic structure of **10** most frequent trace types



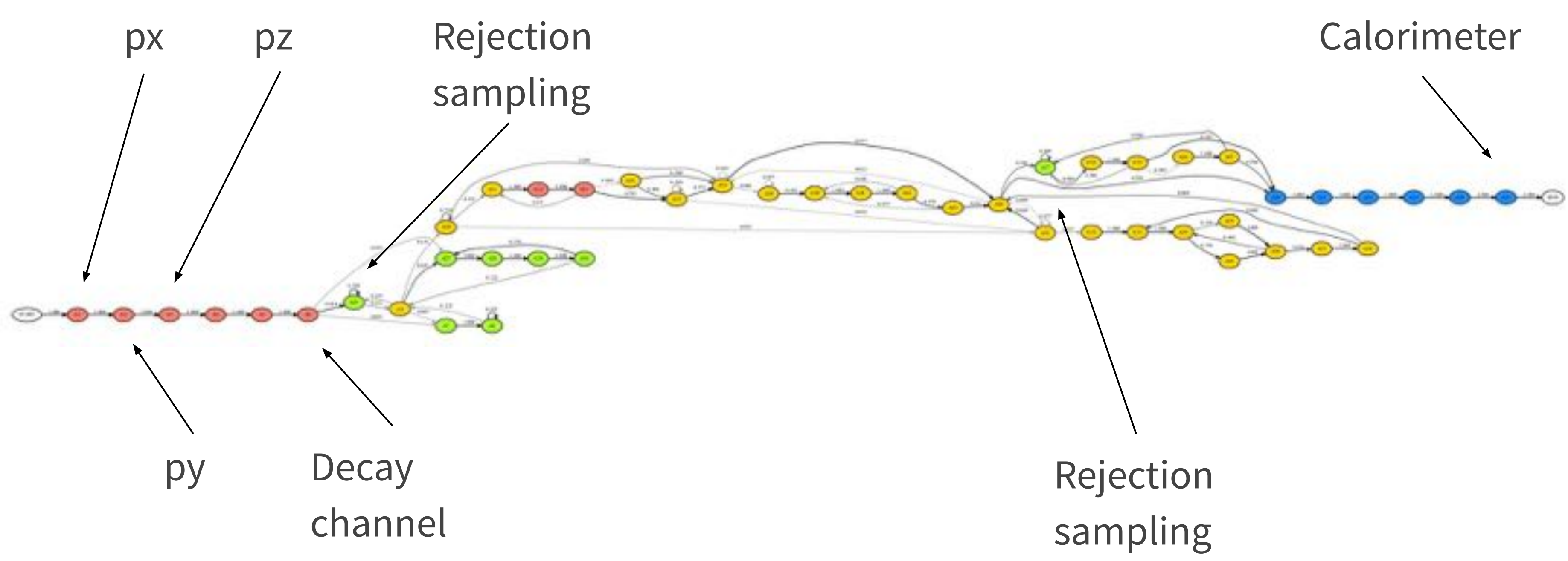
Interpretability

Latent probabilistic structure of **25** most frequent trace types



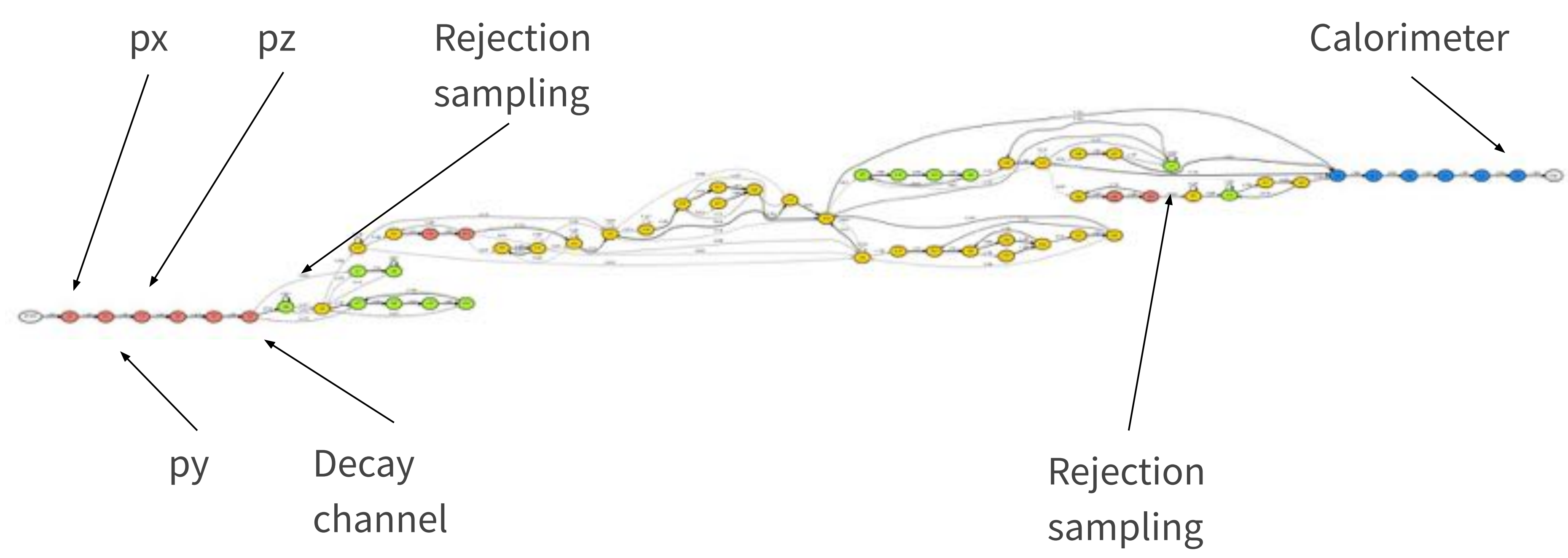
Interpretability

Latent probabilistic structure of **100** most frequent trace types



Interpretability

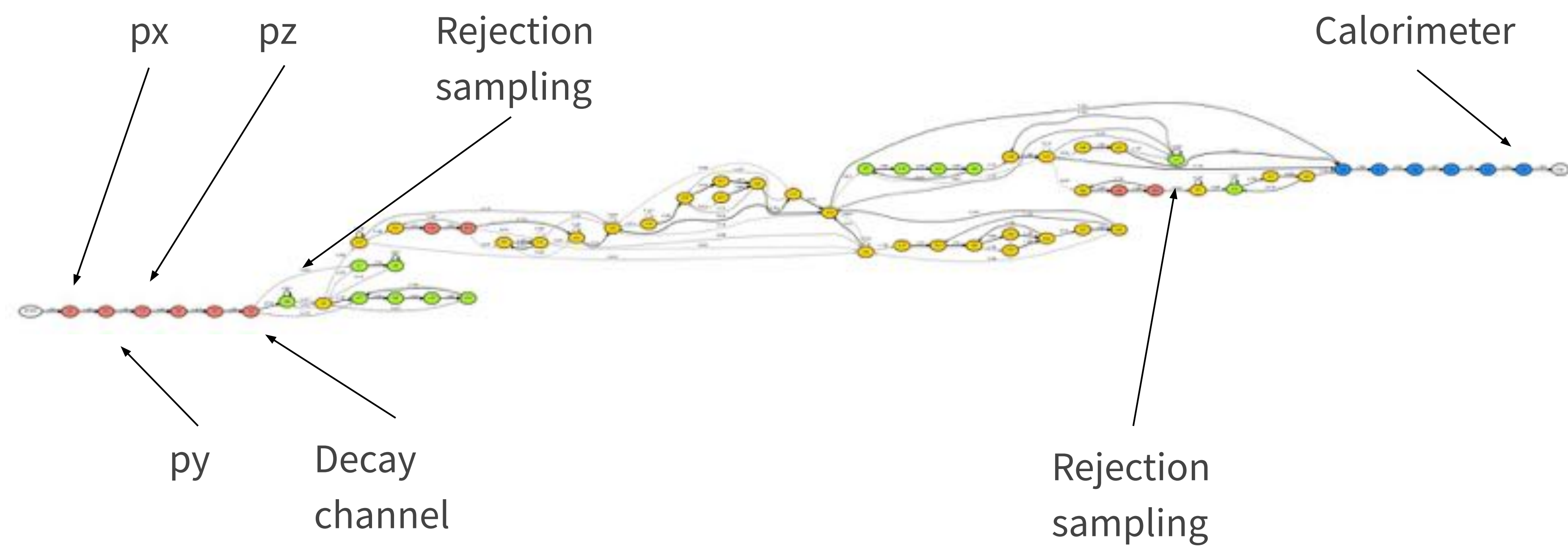
Latent probabilistic structure of **250** most frequent trace types



Probabilistic surrogate models

Probabilistic surrogate networks

Can we **replace the simulator** entirely with a **surrogate model with the same address structure**?



Munk, Zwartsenberg, Ścibior, **Baydin**, Stewart, Fernlund, Poursartip, Wood. 2022. "**Probabilistic Surrogate Networks for Simulators with Unbounded Randomness**" In 38th Conference on Uncertainty in Artificial Intelligence **UAI 2022**

Probabilistic surrogate networks

Demonstrated with a simulation of composite material heating cycles

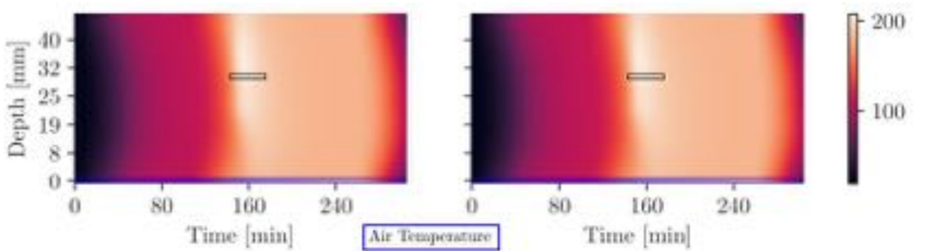
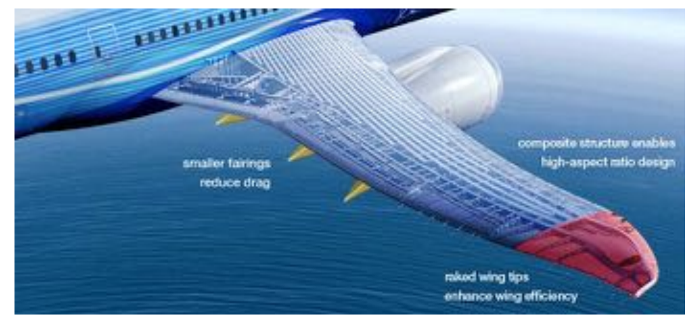


Figure 1: Illustration of a process simulation of composite materials, which we denote x_{RAVEN}. Each subfigure shows a temperature profile measured in degrees Celsius as a function of time along the x axis and depth along the y-axis. (Left) shows the actual simulated heating process. (Right) shows the same process but originating from our *probabilistic surrogate network* trained to match the original simulator. Noticeably, the process in (right) is **26.8 times faster** than the process in (left). Later we perform inference in this process, where we seek to infer the expected temperature in the time window [155, 165] min at depth 30 mm represented by the black box conditioned on measuring the temperature at the bottom surface and air temperature represented by the blue boxes.



Munk, Zwartsenberg, Ścibior, **Baydin**, Stewart, Fernlund, Poursartip, Wood. 2022. "**Probabilistic Surrogate Networks for Simulators with Unbounded Randomness**" In 38th Conference on Uncertainty in Artificial Intelligence **UAI 2022**

Generative Model

```
def gaussian_mixture(x):
    d1 = Categorical(prob=[1/2, 1/2])
    c = sample(dist=d1, address=α1)

    if c==0:
        d2 = Normal(mean=1, std=1)
        m = sample(dist=d2, address=α2)
    else c==1:
        d2 = Normal(mean=-1, std=1)
        m = sample(dist=d2, address=α3)

    d3 = Normal(mean=m, std=1)
    observe(x, likelihood=d3, address=α4)

    return c
```

Inference Network

Sampled

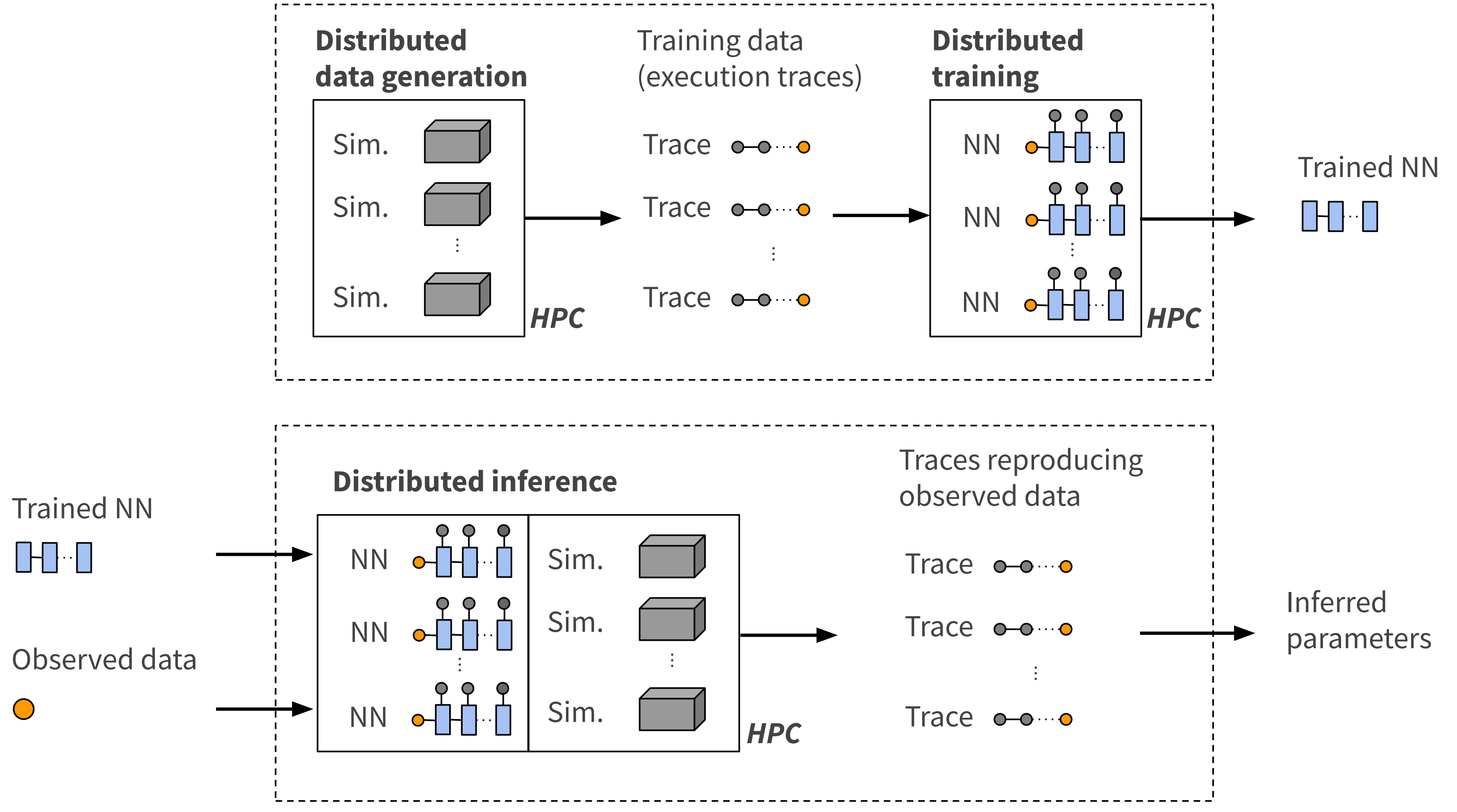
LSTM

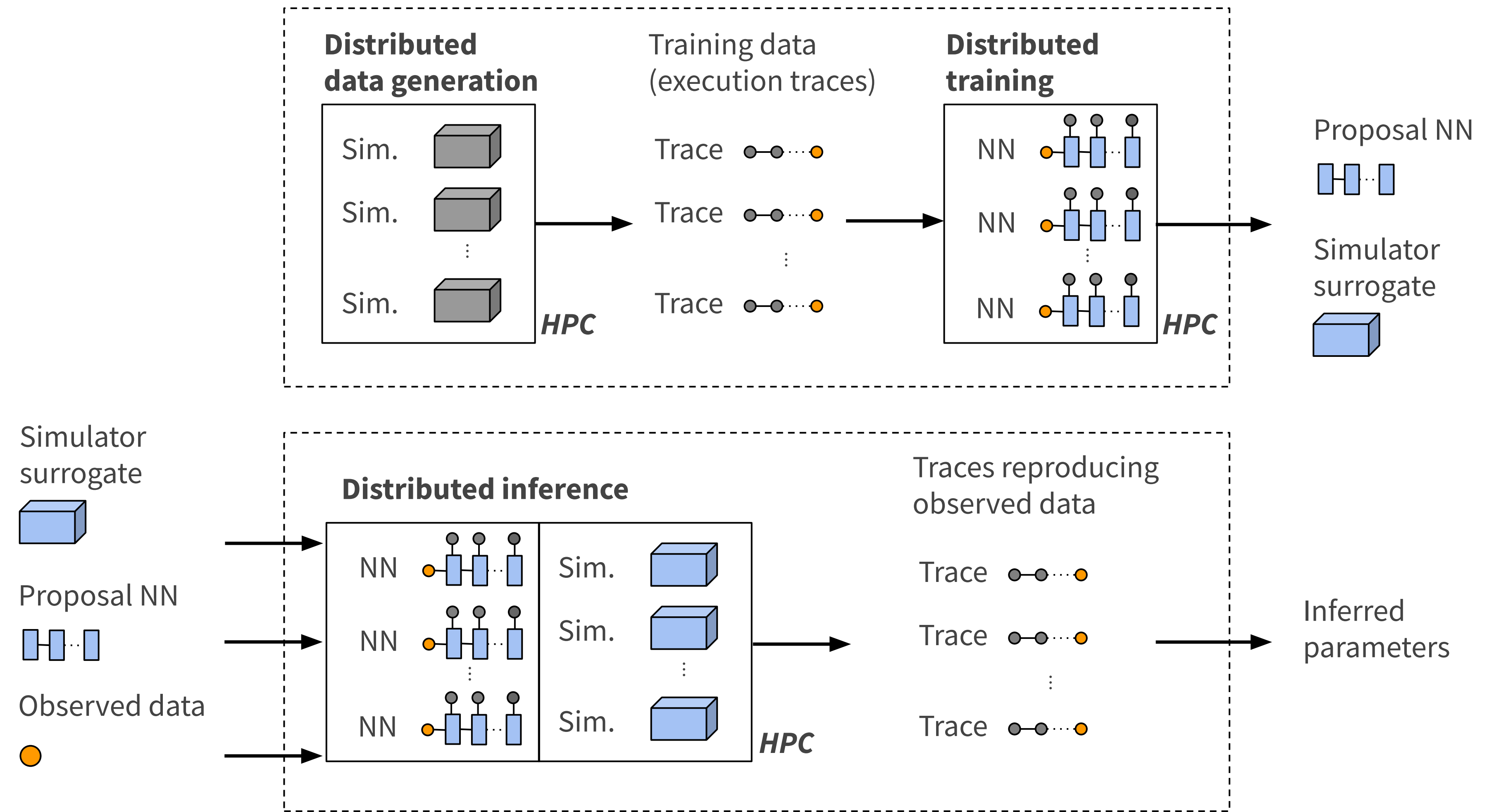
Probabilistic Surrogate Network

LSTM

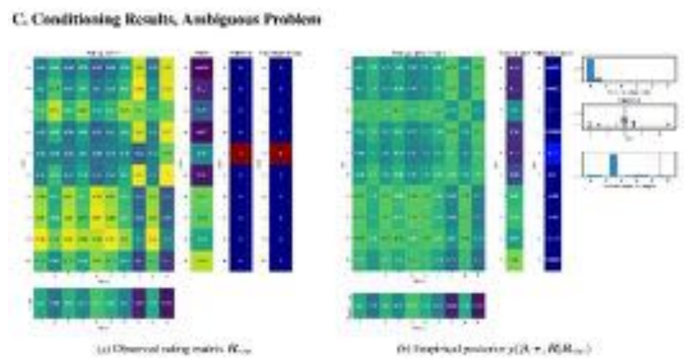
Sampled

Munk, Zwartsenberg, Ścibior, **Baydin**, Stewart, Fernlund, Poursartip, Wood. 2022. “**Probabilistic Surrogate Networks for Simulators with Unbounded Randomness**” In 38th Conference on Uncertainty in Artificial Intelligence **UAI 2022**


Distributed data generation
Sim.
Sim.
Sim.
HPC
Training data (execution traces)
Trace
Trace
Trace
Distributed training
NN
NN
NN
HPC
Trained NN
Distributed inference
Trained NN
Observed data
NN
NN
NN
Sim.
Sim.
Sim.
HPC
Traces reproducing observed data
Trace
Trace
Trace
Inferred parameters


Distributed
data generation
Sim.
Sim.
Sim.
HPC
Training data
(execution traces)
Trace
Trace
Trace
Distributed
training
NN
NN
NN
HPC
Proposal NN
Simulator
surrogate
Simulator
surrogate
Proposal NN
Observed data
Distributed inference
NN
NN
NN
Sim.
Sim.
Sim.
HPC
Traces reproducing
observed data
Trace
Trace
Trace
Inferred
parameters

Other simulators as probabilistic programs



Spacecraft collision avoidance

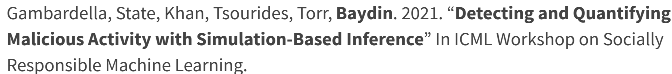
Collaboration with **European Space Agency (ESA)**

Acciarini, Pinto, Metz, Boufelja, Kaczmarek, Merz, Martinez-Heras, Letizia, Bridges, **Baydin**. 2021. “**Kessler: a Machine Learning Library for Space Collision Avoidance.**” In 8th European Conference on Space Debris.

Pinto, Acciarini, Metz, Boufelja, Kaczmarek, Merz, Martinez-Heras, Letizia, Bridges, **Baydin**. 2020. “**Towards Automated Satellite Conjunction Management with Bayesian Deep Learning.**” In AI for Earth Sciences Workshop at NeurIPS 2020, Vancouver, Canada

Social media models and simulation of misinformation dynamics

Collaboration with NYU Center for Social Media and Politics, Oxford Internet Institute



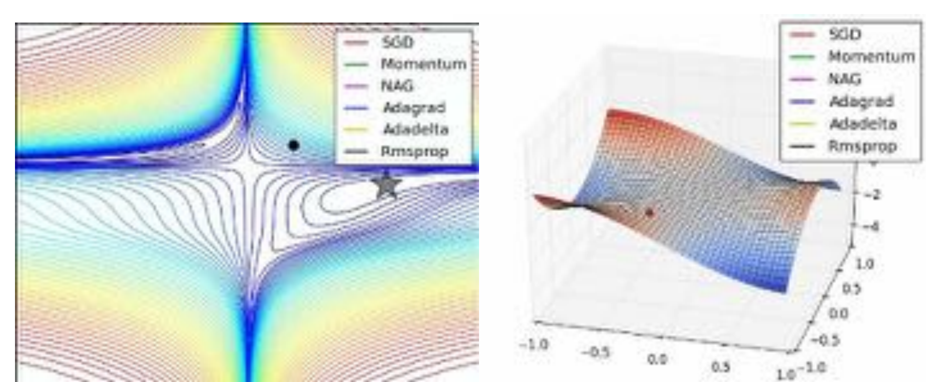
Gambardella, State, Khan, Tsourides, Torr, **Baydin**. 2021. “**Detecting and Quantifying Malicious Activity with Simulation-Based Inference**” In ICML Workshop on Socially Responsible Machine Learning.

Mehta, State, Bonneau, Nagler, Torr, **Baydin**. 2022 “**Estimating the Impact of Coordinated Inauthentic Behavior on Content Recommendations in Social Networks**” ICML workshop on AI for Agent Based Modeling

Differentiable programming and simulators

What is differentiable programming?

Deep learning is behind many recent advances in machine learning
= nonlinear **differentiable functions (programs)**
whose parameters are tuned by **gradient-based optimization**

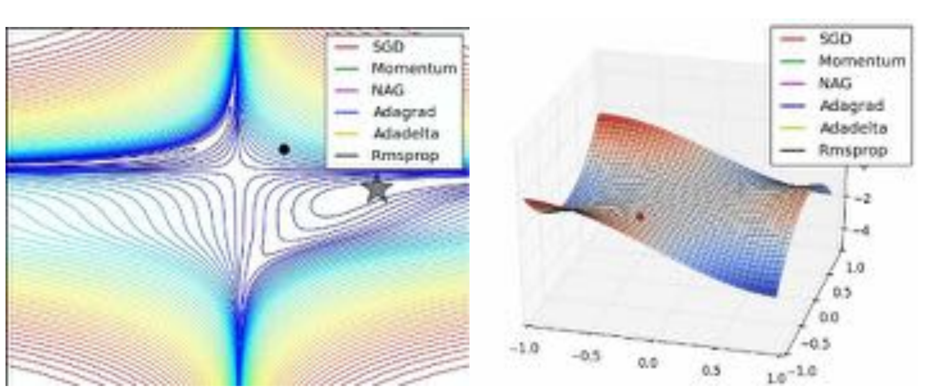


(Ruder, 2017) http://ruder.io/optimizing-gradient-descent/

What is differentiable programming?

Deep learning is behind many recent advances in machine learning
= nonlinear **differentiable functions (programs)**
whose parameters are tuned by **gradient-based optimization**

Derivatives come from running the code via **automatic differentiation**
(mainly backpropagation / reverse mode)



(Ruder, 2017) http://ruder.io/optimizing-gradient-descent/

$$f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$$

↓ **automatic differentiation**

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

Differentiable programming

Differentiable programming is a generalization of deep learning

- Write programs composed of **differentiable and parameterized building blocks** executed via **automatic differentiation**
- **Optimize parameters** in order to perform a specified task, based on data

(2018)

OK, Deep Learning has outlived its usefulness as a buzz-phrase.
Deep Learning est mort. Vive Differentiable Programming!

Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers.

Andrej Karpathy (2017)
"Software 2.0"
https://karpathy.medium.com/software-2-0-a64152b37c35

Differentiable programming

Differentiable programming is a generalization of deep learning

- Write programs composed of **differentiable and parameterized building blocks** executed via **automatic differentiation**
- **Optimize parameters** in order to perform a specified task, based on data

(Neural networks are just a simple class of differentiable functions)

OK, Deep Learning has outlived its usefulness as a buzz-phrase.
Deep Learning est mort. Vive Differentiable Programming!

Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers.

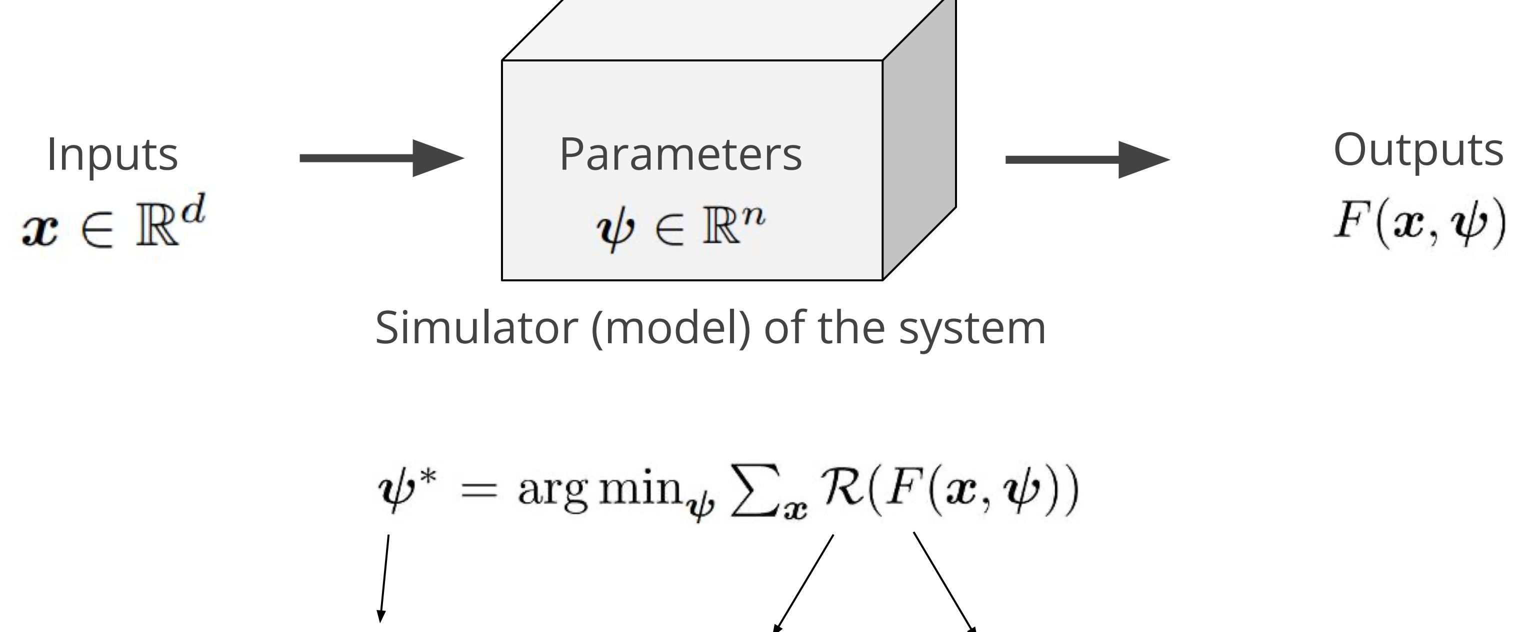
Andrej Karpathy (2017)
"Software 2.0"
https://karpathy.medium.com/software-2-0-a64152b37c35

Simulators and differentiability

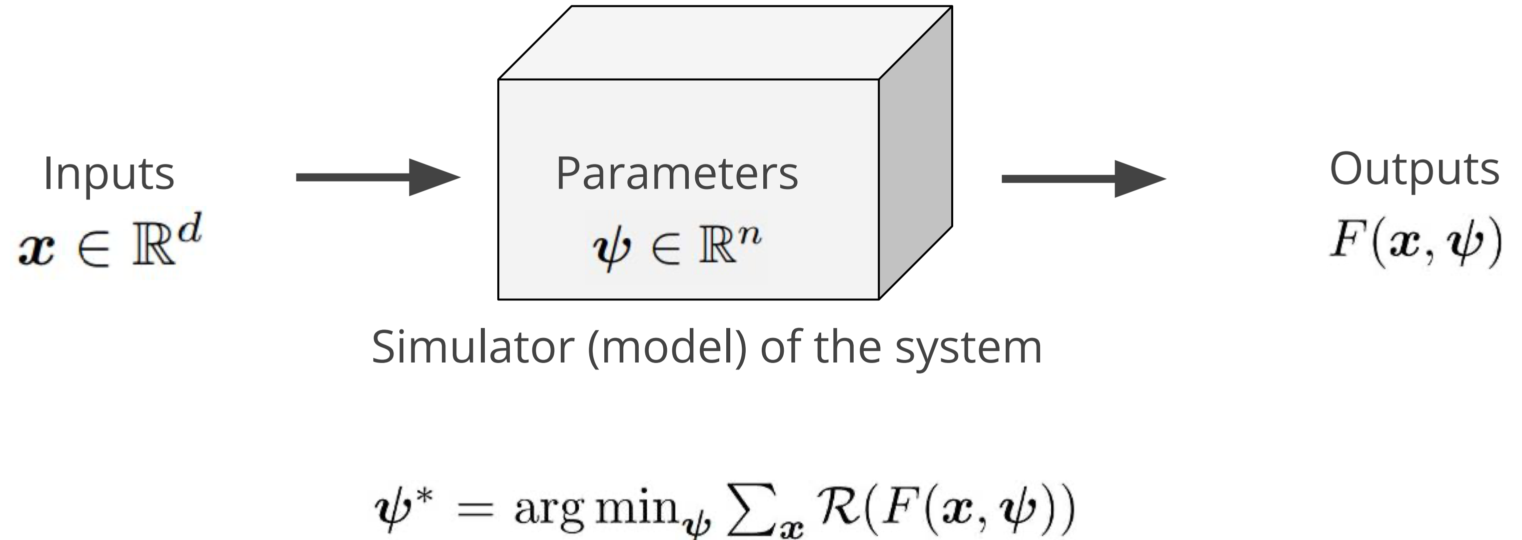
- Simulator code is **not differentiable**
  □ Use surrogates (differentiable approximation learned from data)

- Simulator code is **differentiable**
  (but has not been used in a differentiable way so far)
  □ Use automatic differentiation if feasible

Simulators and differentiability

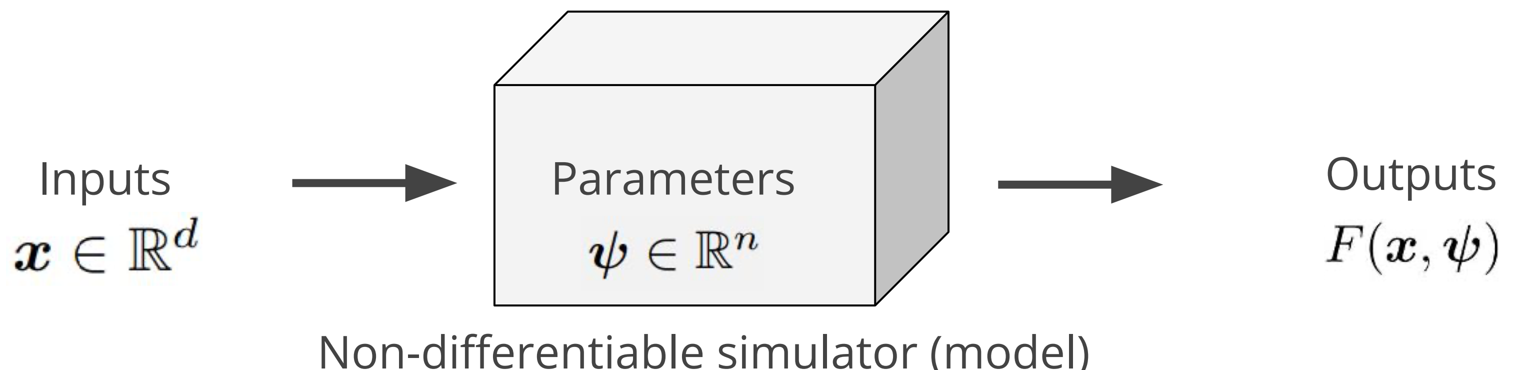


Simulators and differentiability



Can be efficiently found by gradient-based optimization if $\nabla_{\boldsymbol{\psi}} \mathcal{R}$ is available

Non-differentiable simulator

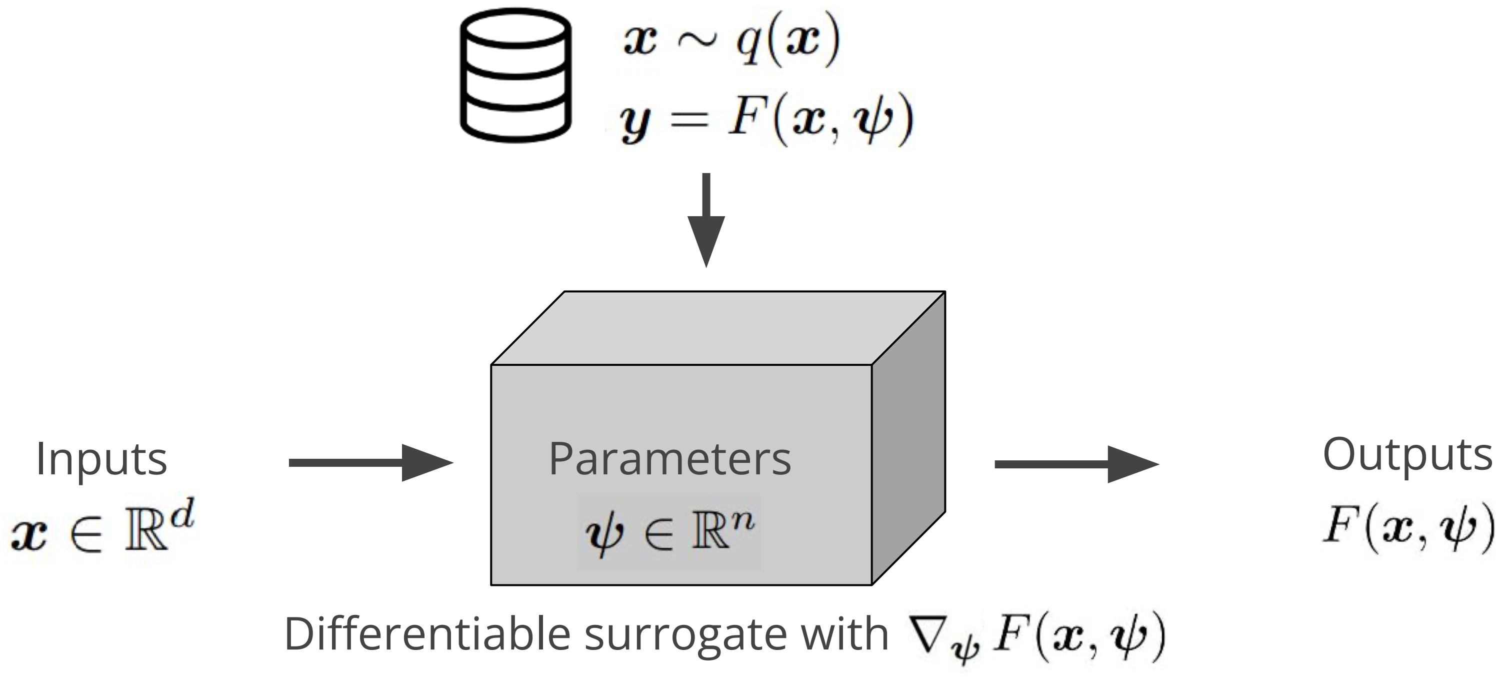


- Run simulator many times
- Generate a (large) dataset of input - output pairs capturing simulator's behavior

$$\boldsymbol{x} \sim q(\boldsymbol{x})$$
$$\boldsymbol{y} = F(\boldsymbol{x}, \boldsymbol{\psi})$$

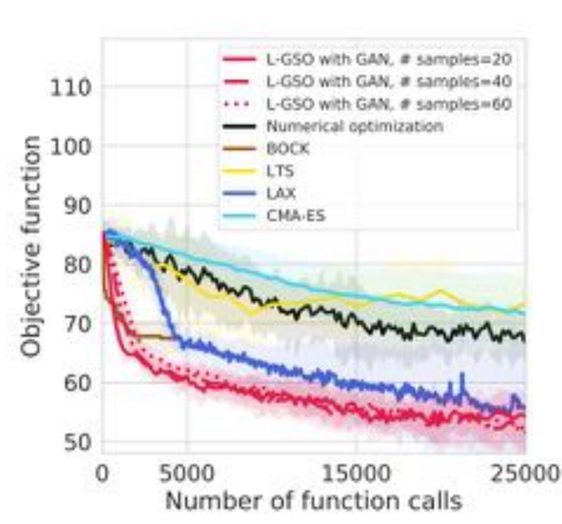
Non-differentiable simulator

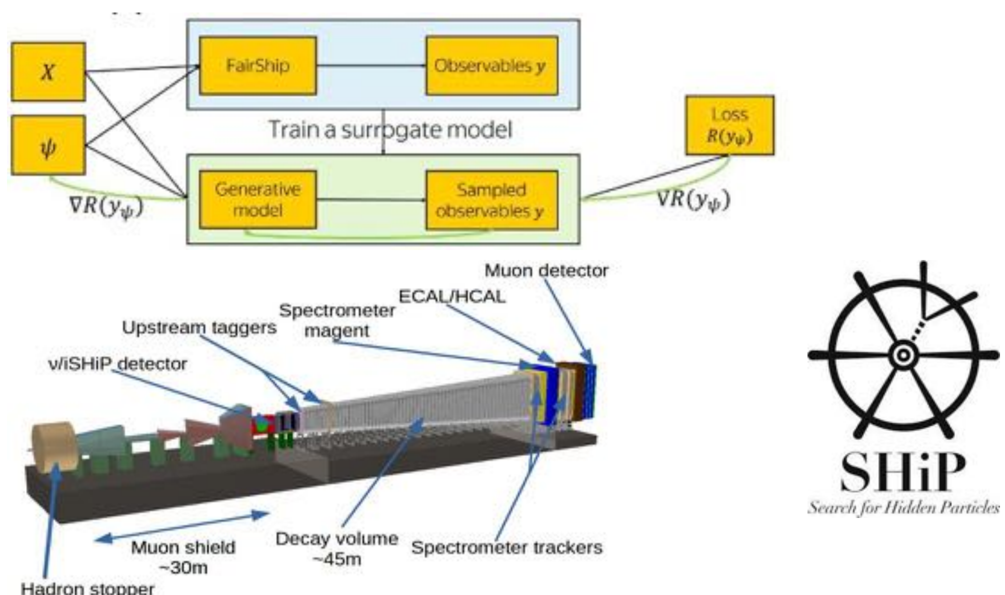
- Use the dataset to **learn a surrogate** (differentiable approximation) of the simulator (e.g., a deep generative model)



Example: local generative surrogates

- Deep generative surrogates (GAN) trained in successive local neighborhoods
- Optimize SHiP muon shield (GEANT4, FairRoot), minimize number of recorded muons by varying magnet geometry

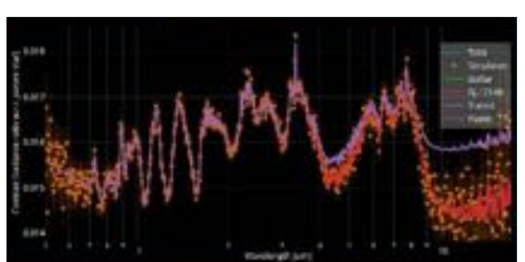


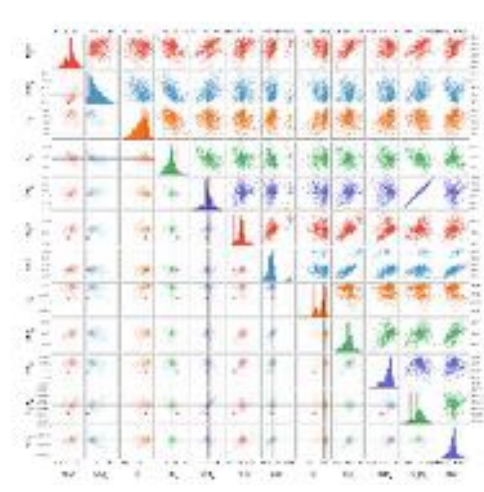


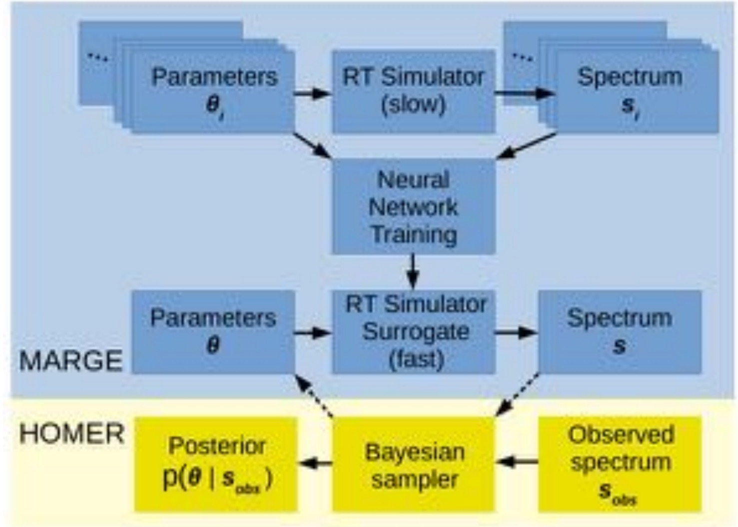
Shirobokov, Belavin, Kagan, Ustyuzhanin, ***Baydin "Black-Box Optimization with Local Generative Surrogates" NeurIPS 2020***

Example: exoplanet radiative transfer

- **Posterior distributions of gas concentrations** in exoplanet atmospheres, conditioned on observed spectra, using radiative transfer simulators
- Surrogates allow up to 180x faster inference



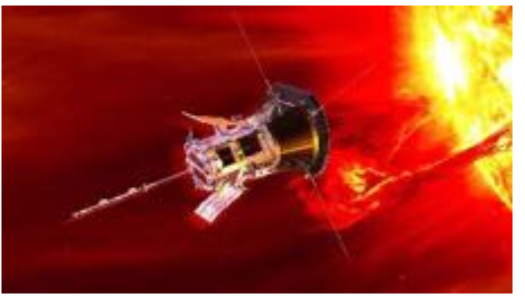




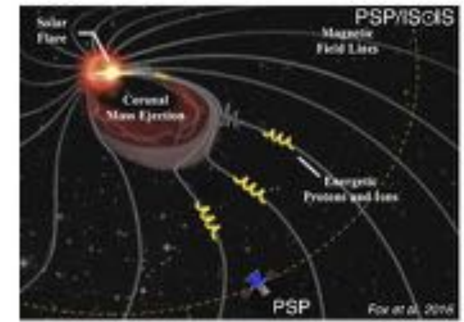
*Himes, Harrington, Cobb, **Baydin**, Soboczenski, O'Beirne, Zorzan, Wright, Scheffer, Domagal-Goldman, Arney. 2022. "**Accurate Machine-Learning Atmospheric Retrieval via a Neural-Network Surrogate Model for Radiative Transfer**" The Planetary Science Journal 3 (4). American Astronomical Society: 236–250. doi:10.3847/PSJ/abe3fd.*

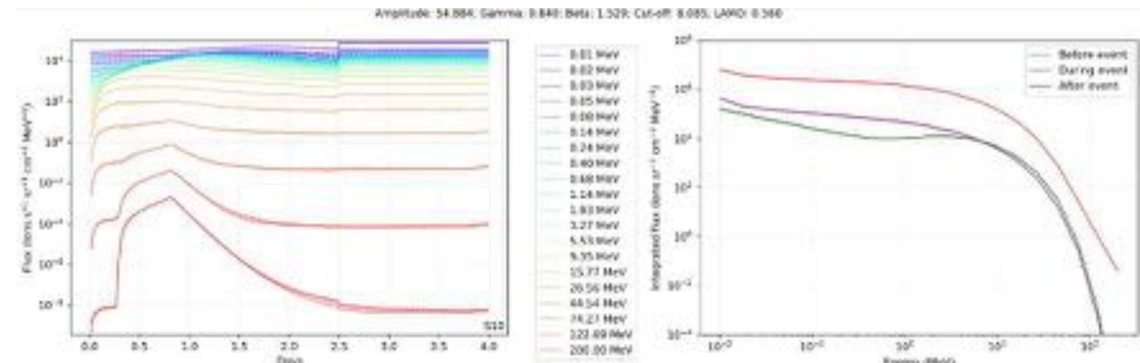
Example: solar energetic particles

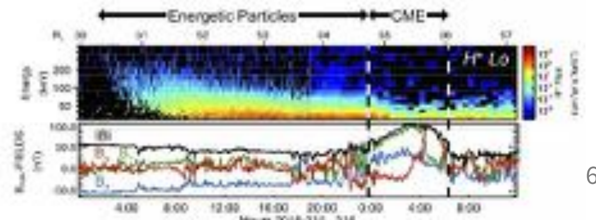
- Solar energetic particles (SEPs) pose serious threats to humans in deep space exploration and to the scientific instruments onboard spacecraft
- Develop an EPREM simulator surrogate (~ billion times faster) enabling posterior inference of seed particle populations **conditioned on real space weather events**



NASA Parker Solar Probe

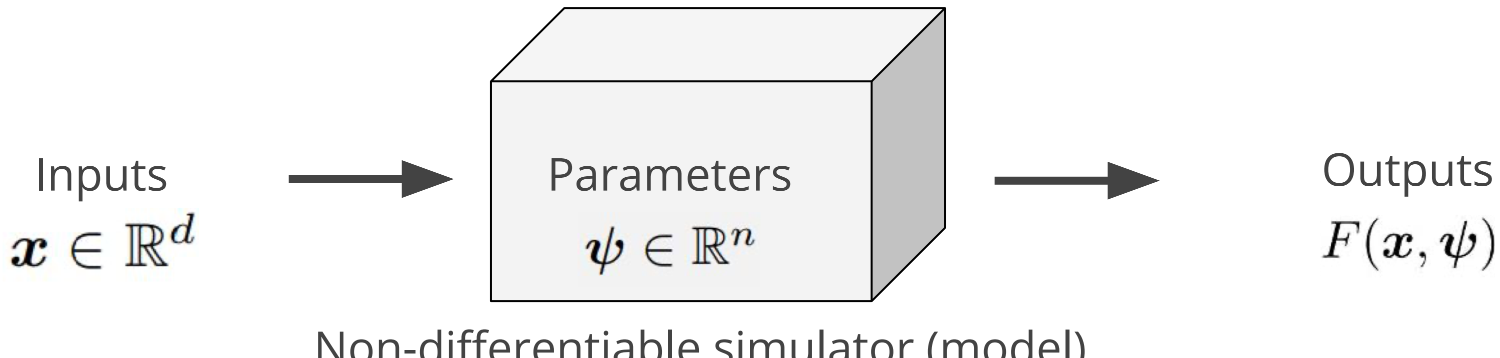




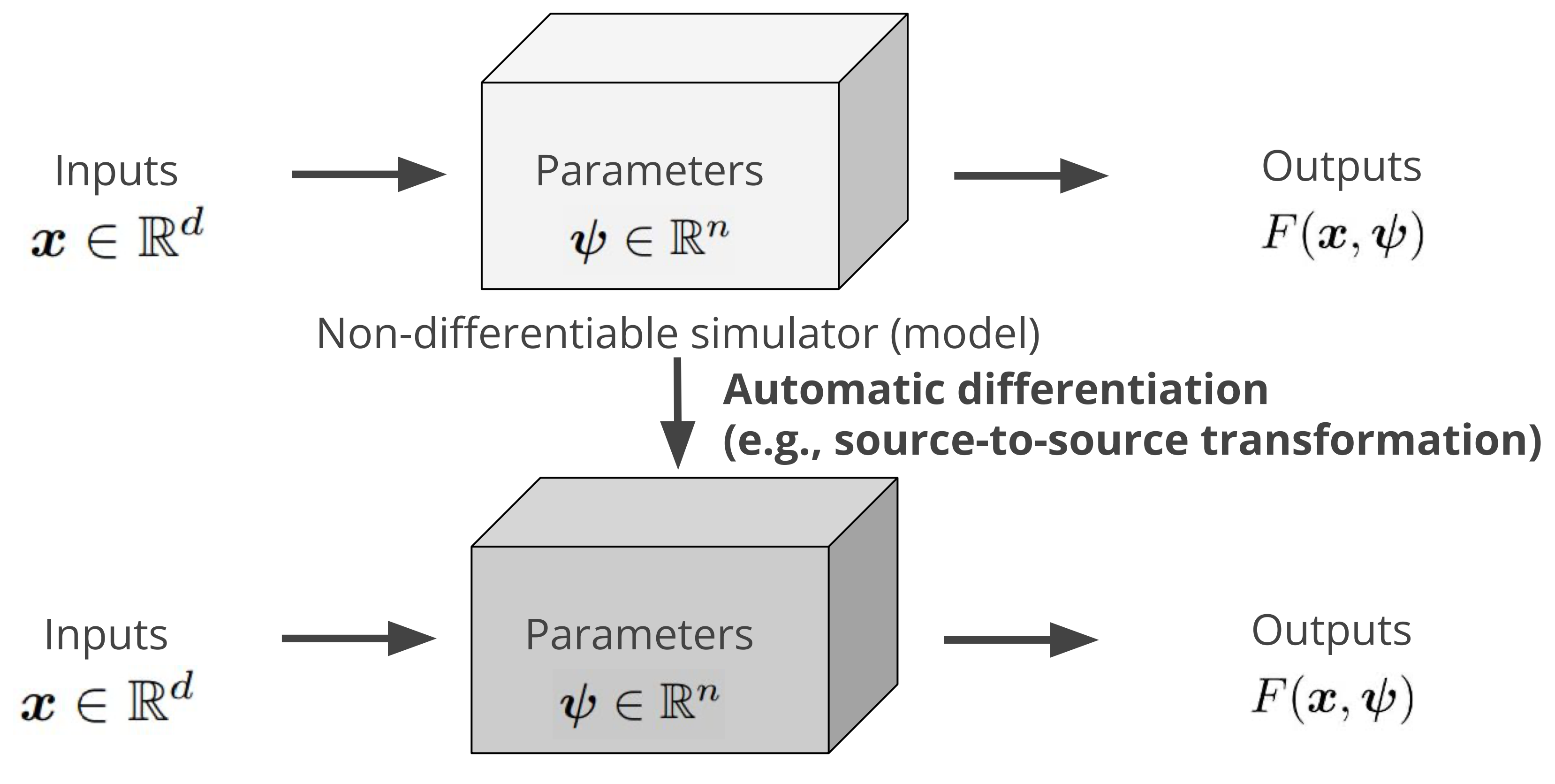


*Poduval, **Baydin**, Schwadron. 2021. "**Studying Solar Energetic Particles and Their Seed Population Using Surrogate Models**" In Machine Learning for Space Sciences Workshop, 43rd Committee on Space Research (COSPAR) Scientific Assembly, Sydney, Australia.*

Differentiable simulator



Differentiable simulator



Differentiable simulator

- Use automatic differentiation tools to make the simulator directly differentiable
- Used in design optimization by the AD community for **many decades**

17 Aerofoil Optimisation via AD of a Multigrid Cell-Vertex Euler Flow Solver

Shaun A. Forth and Trevor P. Evans

ABSTRACT We report preliminary results in the use of ADIFOR 2.0 to determine aerodynamic sensitivities of a 2-D airfoil with respect to geometrical variables. Meshes are produced with a hyperbolic interpolation technique. The flow field is calculated using the cell-vertex method of Hall, which incorporates local time-stepping, mesh sequencing and multigrid. We present results and timings using both Finite Differences (FD) and Automatic Differentiation (AD). We investigate the effect of starting the perturbed calculation for FD and the derivative calculation for AD from either the current or freestream conditions and highlight the need for careful implementation of convergence criteria.
We attempt to make a comparative study of AD and FD gradients in an aerofoil optimisation, using the DERA CODAS method from the perspective of DERA's eventual aim, 3D viscous optimisation of wing-body configurations.

Application of Automatic Differentiation to Race Car Performance Optimisation

Daniele Casanova, Robin S. Sharp, Mark Final, Bruce Christianson and Pat Symonds

ABSTRACT A formal method for the evaluation of the minimum time vehicle manoeuvre is described. The problem is treated as one of optimal control and is solved using a direct transcription method. The resulting nonlinear programming problem is solved using the sequential quadratic programming algorithm SNOPT for constrained optimisation. The automatic differentiation software tool $\mathbf{AD}_{opt}$ is used for the evaluation of the first-order derivatives of objective and constraint functions with respect to the control variables. The implementation of automatic differentiation is more robust and ten times as fast compared to the use of a finite difference determination of the Jacobian.

Forth, Shaun A.; Evans, Trevor P. Aerofoil Optimisation via AD of a Multigrid Cell-Vertex Euler Flow Solver. 2002

Daniele Casanova, Robin S. Sharp, Mark Final, Bruce Christianson, Pat Symonds. “Application of Automatic Differentiation to Race Car Performance Optimisation” in Automatic Differentiation of Algorithms: From Simulation to Optimization, Springer, 2002

End-to-end differentiable pipelines

- Complex experimental setups can be composed of a pipeline of a series of distinct simulators (e.g., SHERPA -> GEANT)
- One might need to differentiate through the whole end-to-end pipeline, which can be achieved by compositionality and the chain rule

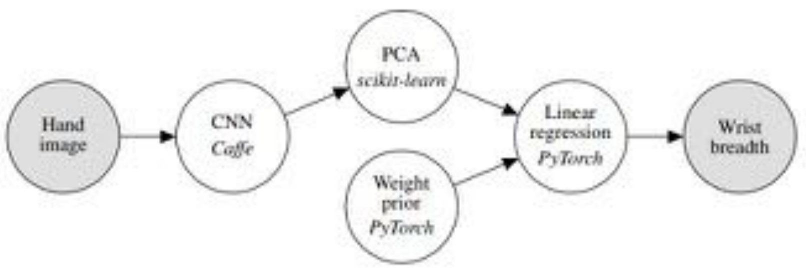


Figure 2: Example pipeline for predicting a physiological feature (wrist breadth) from images of hands using four primitives in three different frameworks.

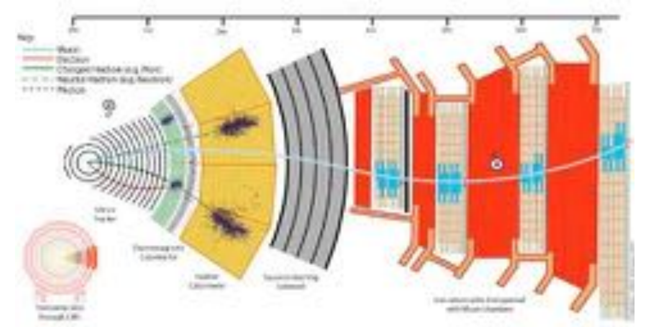
DARPA Data Driven Discovery of Models (D3M)
https://datadrivendiscovery.org/

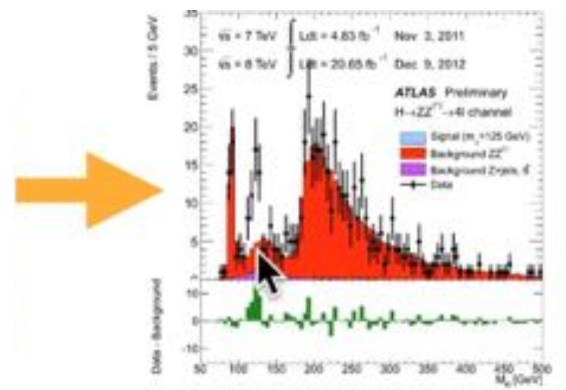
Milutinovic, **Baydin**, Zinkov, Harvey, Song, Wood, Shen. 2017. "End-to-End Training of Differentiable Pipelines Across Machine Learning Frameworks." NeurIPS 2017 Autodiff Workshop

Differentiable programming in particle physics

- **Differentiable analysis pipelines**
  Unify analysis pipeline by simultaneously optimizing the free parameters of an analysis with respect to the desired physics objective

- **Gradient-based inference (probabilistic programming)**
  Enable efficient simulation-based inference, reducing the number of events needed by orders of magnitude





Baydin, Cranmer, Feickert, Gray, Heinrich, Held, Melo, Neubauer, Pearkes, Simpson, Smith, Stark, Thais, Vassilev, Watts. 2020. “Differentiable Programming in High-Energy Physics.” In Snowmass 2021 Letters of Interest (LOI), Division of Particles and Fields (DPF), American Physical Society. https://snowmass21.org/loi.

End-to-end differentiable pipelines

- Complex experimental setups can be composed of a pipeline of a series of distinct simulators (e.g., SHERPA -> GEANT)
- One might need to differentiate through the whole end-to-end pipeline, which can be achieved by compositionality and the chain rule

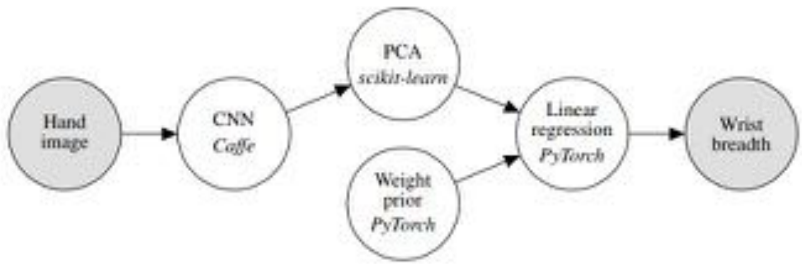


Figure 2: Example pipeline for predicting a physiological feature (wrist breadth) from images of hands using four primitives in three different frameworks.

DARPA Data Driven Discovery of Models (D3M)
https://datadrivendiscovery.org/

Milutinovic, **Baydin**, Zinkov, Harvey, Song, Wood, Shen. 2017. “End-to-End Training of Differentiable Pipelines Across Machine Learning Frameworks.” NeurIPS 2017 Autodiff Workshop

DiffSharp

DiffSharp

- An autodiff library in F#
- Supporting arbitrary nesting of forward/reverse AD
- a higher-order differentiation API
- Based on Barak Pearlmutter and Jeffrey Siskind's functional AD work (e.g., R6RS-AD)

https://diffsharp.github.io/

	Op.	Value	Type signature	AD
$f : \mathbb{R} \to \mathbb{R}$	`diff`	f'	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R}$	X, F
	`diff'`	(f, f')	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to (\mathbb{R} \times \mathbb{R})$	X, F
	`diff2`	f''	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R}$	X, F
	`diff2'`	(f, f'')	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to (\mathbb{R} \times \mathbb{R})$	X, F
	`diff2''`	(f, f', f'')	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to (\mathbb{R} \times \mathbb{R} \times \mathbb{R})$	X, F
	`diffn`	$f^{(n)}$	$\mathbb{N} \to (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R}$	X, F
	`diffn'`	$(f, f^{(n)})$	$\mathbb{N} \to (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to (\mathbb{R} \times \mathbb{R})$	X, F
$f : \mathbb{R}^n \to \mathbb{R}$	`grad`	∇f	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n$	X, R
	`grad'`	$(f, \nabla f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n)$	X, R
	`gradv`	$\nabla f \cdot \mathbf{v}$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}$	X, F
	`gradv'`	$(f, \nabla f \cdot \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R})$	X, F
	`hessian`	$\mathbf{H}_f$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^{n \times n}$	X, R-F
	`hessian'`	$(f, \mathbf{H}_f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^{n \times n})$	X, R-F
	`hessianv`	$\mathbf{H}_f \mathbf{v}$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}^n$	X, F-R
	`hessianv'`	$(f, \mathbf{H}_f \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n)$	X, F-R
	`gradhessian`	$(\nabla f, \mathbf{H}_f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R}^n \times \mathbb{R}^{n \times n})$	X, R-F
	`gradhessian'`	$(f, \nabla f, \mathbf{H}_f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n \times n})$	X, R-F
	`gradhessianv`	$(\nabla f \cdot \mathbf{v}, \mathbf{H}_f \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n)$	X, F-R
	`gradhessianv'`	$(f, \nabla f \cdot \mathbf{v}, \mathbf{H}_f \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R} \times \mathbb{R}^n)$	X, F-R
	`laplacian`	$\mathrm{tr}(\mathbf{H}_f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}$	X, R-F
	`laplacian'`	$(f, \mathrm{tr}(\mathbf{H}_f))$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R})$	X, R-F
$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$	`jacobian`	$\mathbf{J}_\mathbf{f}$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^{m \times n}$	X, F/R
	`jacobian'`	$(\mathbf{f}, \mathbf{J}_\mathbf{f})$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to (\mathbb{R}^m \times \mathbb{R}^{m \times n})$	X, F/R
	`jacobianv`	$\mathbf{J}_\mathbf{f} \mathbf{v}$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}^m$	X, F
	`jacobianv'`	$(\mathbf{f}, \mathbf{J}_\mathbf{f} \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R}^m \times \mathbb{R}^m)$	X, F
	`jacobianT`	$\mathbf{J}_\mathbf{f}^T$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^{n \times m}$	X, F/R
	`jacobianT'`	$(\mathbf{f}, \mathbf{J}_\mathbf{f}^T)$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to (\mathbb{R}^m \times \mathbb{R}^{n \times m})$	X, F/R
	`jacobianTv`	$\mathbf{J}_\mathbf{f}^T \mathbf{v}$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^m \to \mathbb{R}^n$	X, R
	`jacobianTv'`	$(\mathbf{f}, \mathbf{J}_\mathbf{f}^T \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^m \to (\mathbb{R}^m \times \mathbb{R}^n)$	X, R
	`jacobianTv''`	$(\mathbf{f}, \mathbf{J}_\mathbf{f}^T (\cdot))$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to (\mathbb{R}^m \times (\mathbb{R}^m \to \mathbb{R}^n))$	X, R
	`curl`	$\nabla \times \mathbf{f}$	$(\mathbb{R}^3 \to \mathbb{R}^3) \to \mathbb{R}^3 \to \mathbb{R}^3$	X, F
	`curl'`	$(\mathbf{f}, \nabla \times \mathbf{f})$	$(\mathbb{R}^3 \to \mathbb{R}^3) \to \mathbb{R}^3 \to (\mathbb{R}^3 \times \mathbb{R}^3)$	X, F
	`div`	$\nabla \cdot \mathbf{f}$	$(\mathbb{R}^n \to \mathbb{R}^n) \to \mathbb{R}^n \to \mathbb{R}$	X, F
	`div'`	$(\mathbf{f}, \nabla \cdot \mathbf{f})$	$(\mathbb{R}^n \to \mathbb{R}^n) \to \mathbb{R}^n \to (\mathbb{R}^n \times \mathbb{R})$	X, F
	`curldiv`	$(\nabla \times \mathbf{f}, \nabla \cdot \mathbf{f})$	$(\mathbb{R}^3 \to \mathbb{R}^3) \to \mathbb{R}^3 \to (\mathbb{R}^3 \times \mathbb{R})$	X, F
	`curldiv'`	$(\mathbf{f}, \nabla \times \mathbf{f}, \nabla \cdot \mathbf{f})$	$(\mathbb{R}^3 \to \mathbb{R}^3) \to \mathbb{R}^3 \to (\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R})$	X, F

DiffSharp 1.0

- Major reimplementation with Don Syme (Microsoft/GitHub Next)
- Introduce tensor backends, default being libtorch (PyTorch's core in C++ / CUDA)
- Uses libtorch tensors without autograd, implements its own computation graph and differentiation code
- F# notebooks in Jupyter, VS Code
- Supports MacOS, Linux, Windows, CUDA

```
let encoder =
    Conv2d(1, 32, 4, 2)
    --> dsharp.relu
    --> Conv2d(32, 64, 4, 2)
    --> dsharp.relu
    --> Conv2d(64, 128, 4, 2)
    --> dsharp.flatten(1)

let decoder =
    dsharp.unflatten(1, [128;1;1])
    --> ConvTranspose2d(128, 64, 4, 2)
    --> dsharp.relu
    --> ConvTranspose2d(64, 32, 4, 3)
    --> dsharp.relu
    --> ConvTranspose2d(32, 1, 4, 2)
    --> dsharp.sigmoid

let model = VAE([1;28;28], 64, encoder, decoder)

printfn "Model\n%s" (model.summary())

let optimizer = Adam(model, lr=dsharp.tensor(0.001))
```

```
!bash <(curl -Ls https://raw.githubusercontent.com/gbaydin/scripts/main/colab_dotnet5.sh)
```

DiffSharp 1.0

Can be the basis for your new probabilistic programming systems!

Currently working on a PyProb analog in F# with support for PPX

```
type GaussianUnknownMean() =
    inherit Model()
    override _.forward() =
        let mu = dsharp.sample(Normal(dsharp.tensor(1.),
                                      dsharp.tensor(0.1)), "mu")
        ()

let model = GaussianUnknownMean()

let numTraces = 1000
let prior = model.prior(numTraces)
let priorMu = prior.map (fun trace -> trace.["mu"].value)
let priorMuMean = priorMu.mean
let priorMuMeanCorrect = dsharp.tensor(1.)

print priorMuMean
print priorMuMeanCorrect
Assert.True(priorMuMeanCorrect.allclose(priorMuMean, 0.1))
```

Joining the community

Machine-learning Optimized Design of Experiments (MODE)

Collaboration of ML and physics people at CERN, Padova, UC Louvain, Liege, Oxford, NYU, Rutgers, Uppsala, TU-Munchen, Durham and other places

Use **differentiable programming** in design optimization of **detectors for particle physics**, industrial applications

https://mode-collaboration.github.io/

Machine-learning Optimized Design of Experiments (MODE)

MODE Workshop series on Differentiable Programming for Experiment design

- 6-8 Sep 2021: Université catholique de Louvain, Belgium
- 12-16 Sep 2022: Orthodox Academy of Crete, Greece
- **Abstract submission deadline: 31 Jul 2022**

https://indico.cern.ch/event/1145124/abstracts/

White papers

AG Baydin, K Cranmer, P de Castro Manzano, C Delaere, D Derkach, J Donini, T Dorigo, A Giammanco, J Kieseler, L Layer, G Louppe, F Ratnikov, G Strong, M Tosi, A Ustyuzhanin, P Vischia, H Yarar. 2021. **“Toward Machine Learning Optimization of Experimental Design.”** Nuclear Physics News 31 (1). Taylor & Francis: 25–28. doi:10.1080/10619127.2021.1881364

AG Baydin, K Cranmer, M Feickert, L Gray, L Heinrich, A Held, A Melo, M Neubauer, J Pearkes, N Simpson, N Smith, G Stark, S Thais, V Vassilev, G Watts. 2020. **“Differentiable Programming in High-Energy Physics.”** In Snowmass 2021 Letters of Interest (LOI), Division of Particles and Fields (DPF), American Physical Society. https://snowmass21.org/loi

MIAPbP program

Munich Institute for Astro-, Particle and BioPhysics,
Technical University of Munich

Month-long program in **Differentiable and Probabilistic Programming for Fundamental Physics**

- Organizers: Lukas Heinrich, Torsten Enßlin, Michael Kagan, Atılım Güneş Baydin, Vassil Vassilev
- Bringing together **probabilistic programming** and **fundamental physics** communities
- Hosted in Munich during 5 - 30 Jun 2023
- Registration deadline: 3 Oct 2022

https://www.munich-iapbp.de/probabilistic-programming

Thank you for listening

Questions?

Selected references

[1] T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming. In International Conference on Artificial Intelligence and Statistics (AISTATS), 2017.
[2] A. Munk, A. Ścibior, A. G. Baydin, A. Stewart, G. Fernlund, A. Poursartip, and F. Wood. Deep probabilistic surrogate networks for universal simulator approximation. In PROBPROG, 2020.
[3] A. G. Baydin, L. Heinrich, W. Bhimji, L. Shao, S. Naderiparizi, A. Munk, J. Liu, B. Gram-Hansen, G. Louppe, L. Meadows, P. Torr, V. Lee, Prabhat, K. Cranmer, and F. Wood. Efficient probabilistic inference in the quest for physics beyond the standard model. In NeurIPS, 2019.
[4] A. G. Baydin, L. Shao, W. Bhimji, L. Heinrich, L. F. Meadows, J. Liu, A. Munk, S. Naderiparizi, B. Gram-Hansen, G. Louppe, M. Ma, X. Zhao, P. Torr, V. Lee, K. Cranmer, Prabhat, and F. Wood. Etalumis: Bringing probabilistic programming to scientific simulators at scale. In SC19, 2019.
[5] K. Cranmer, J. Brehmer, and G. Louppe. The frontier of simulation-based inference. Proceedings of the National Academy of Sciences, 117(48):30055–30062, 2020.
[6] B. Gram-Hansen, C. Schroeder, P. H. Torr, Y. W. Teh, T. Rainforth, and A. G. Baydin. Hijacking malaria simulators with probabilistic programming. In ICML workshop on AI for Social Good, 2019.
[7] B. Gram-Hansen, C. S. de Witt, R. Zinkov, S. Naderiparizi, A. Scibior, A. Munk, F. Wood, M. Ghadiri, P. Torr, Y. W. Teh, A. G. Baydin, and T. Rainforth. Efficient bayesian inference for nested simulators. In AABI, 2019.
[8] B. Poduval, A. G. Baydin, and N. Schwadron. Studying solar energetic particles and their seed population using surrogate models. In MML for Space Sciences workshop, COSPAR, 2021.
[9] G. Acciarini, F. Pinto, S. Metz, S. Boufelja, S. Kaczmarek, K. Merz, J. A. Martinez-Heras, F. Letizia, C. Bridges, and A. G. Baydin. Spacecraft collision risk assessment with probabilistic programming. In ML4PS (NeurIPS 2020), 2020.
[10] F. Pinto, G. Acciarini, S. Metz, S. Boufelja, S. Kaczmarek, K. Merz, J. A. Martinez-Heras, F. Letizia, C. Bridges, and A. G. Baydin. Towards automated satellite conjunction management with bayesian deep learning. In AI for Earth Sciences Workshop (NeurIPS), 2020.
[11] G. Acciarini, F. Pinto, S. Metz, S. Boufelja, S. Kaczmarek, K. Merz, J. A. Martinez-Heras, F. Letizia, C. Bridges, and A. G. Baydin. Kessler: a machine learning library for space collision avoidance. In 8th European Conference on Space Debris, 2021.
[12] S. Shirobokov, V. Belavin, M. Kagan, A. Ustyuzhanin, and A. G. Baydin. Black-box optimization with local generative surrogates. In NeurIPS, 2020.
[13] H. S. Behl, A. G. Baydin, R. Gal, P. H. S. Torr, and V. Vineet. Autosimulate: (quickly) learning synthetic data generation. In 16th European Conference on Computer Vision (ECCV), 2020.
[14] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research (JMLR), 18(153):1–43, 2018.
[15] A. G. Baydin, B. A. Pearlmutter, and J. M. Siskind. DiffSharp: An AD library for .net languages. In 7th International Conference on Algorithmic Differentiation, 2016.
[16] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood. Online learning rate adaptation with hypergradient descent. In ICLR, 2018.
[17] H. Behl, A. G. Baydin, and P. H. Torr. Alpha maml: Adaptive model-agnostic meta-learning. In AutoML (ICML), 2019.
[18] A. G. Baydin, K. Cranmer, M. Feickert, L. Gray, L. Heinrich, A. Held, A. Melo, M. Neubauer, J. Pearkes, N. Simpson, N. Smith, G. Stark, S. Thais, V. Vassilev, and G. Watts. Differentiable programming in high-energy physics. In Snowmass 2021 Letters of Interest (LOI), Division of Particles and Fields (DPF), American Physical Society, 2020.
[19] A. G. Baydin, K. Cranmer, P. de Castro Manzano, C. Delaere, D. Derkach, J. Donini, T. Dorigo, A. Giammanco, J. Kieseler, L. Layer, G. Louppe, F. Ratnikov, G. C. Strong, M. Tosi, A. Ustyuzhanin, P. Vischia, and H. Yarar. Toward machine learning optimization of experimental design. Nuclear Physics News International (Submitted), 2020.
[20] L. F. Guedes dos Santos, S. Bose, V. Salvatelli, B. Neuberg, M. Cheung, M. Janvier, M. Jin, Y. Gal, P. Boerner, and A. G. Baydin. Multi-channel auto-calibration for the atmospheric imaging assembly using machine learning. Astronomy & Astrophysics (in press), 2021.
[21] A. D. Cobb, M. D. Himes, F. Soboczenski, S. Zorzan, M. D. O'Beirne, A. G. Baydin, Y. Gal, S. D. Domagal-Goldman, G. N. Arney, and D. Angerhausen. An ensemble of bayesian neural networks for exoplanetary atmospheric retrieval. The Astronomical Journal, 158(1), 2019.
[22] C. Schroeder de Witt, B. Gram-Hansen, N. Nardelli, A. Gambardella, R. Zinkov, P. Dokania, N. Siddharth, A. B. Espinosa-Gonzalez, A. Darzi, P. Torr, and A. G. Baydin. Simulation-based inference for global health decisions. In ICML Workshop on Machine Learning for Global Health, Thirty-seventh International Conference on Machine Learning (ICML 2020), 2020.